MPC8610 Integrated Host Processor Reference Manual

MPC8610RM
Rev. 1, 06/2010

freescale semiconductor

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 010 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
+1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: MPC8610RM
Rev. 1, 06/2010

Contents

Contents

Chapter 3
Signal Descriptions

Chapter 4
Reset, Clocking, and Initialization

Contents

Contents

Contents

Chapter 6
e600 Core Registers and Instruction Set Summary

Contents

Contents

Contents

Contents

Contents

Contents

Chapter 9
Enhanced Local Bus Controller

Contents

Contents

Contents

Chapter 10
Display Interface Unit (DIU)

Contents

Contents

Chapter 11
Programmable Interrupt Controller (PIC)

Contents

Contents

Contents

Contents

Chapter 13
DUART

Contents

Chapter 14
Fast/Serial Infrared Interfaces (FIRI/SIRI)

Contents

Contents

Chapter 15
Serial Peripheral Interface

Contents

Chapter 16
Synchronous Serial Interface (SSI)

Contents

Chapter 17
Global Timer Module

Contents

Chapter 18
Watchdog Timer

Contents

Contents

Chapter 20
PCI Bus Interface

Contents

Contents

Contents

Chapter 21
PCI Express Interface Controller

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Contents

Appendix A
Complete List of Configuration, Control, and Status Registers

Contents

Appendix B
Revision History

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Figures

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

Tables

About This Book

This reference manual defines the functionality of the MPC8610. The MPC8610 integrates an e600 Power Architecture™ core with the system logic required for general-purpose embedded applications. The e600 processor core is a low-power implementation of the family of reduced instruction set computing (RISC) embedded processors that implement Power Architecture technology. This book is intended as a companion to the *PowerPC e600 Core Complex Reference Manual*.

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:

Part I, "Overview," describes the many features of the MPC8610 integrated host processor at an overview level. The following chapters are included:

- Chapter 1, "MPC8610 Overview," provides a high-level description of features and functionality of the MPC8610 integrated host processor. It describes the MPC8610, its interfaces, and its programming model. The functional operation of the MPC8610 with emphasis on peripheral functions is also described.
- Chapter 2, "Memory Map," describes the memory map of the MPC8610. An overview of the local address map is followed by a description of how local access windows, address translation and mapping units, and the configuration, control, and status register space are used to define the local address map and the configuration, control, and status registers within that map.
- Chapter 3, "Signal Descriptions," provides a listing of all the external signals, cross-references for signals that serve multiple functions, output signal states at reset, and reset configuration signals (and the modes they define).
- 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 MPC8610.

Part II, "e600 Core," describes the many features of the e600 processor core at an overview level and the interaction between the core complex and the L2 cache. The following chapters are included:

- Chapter 5, "e600 Core Overview," provides an overview of the e600 core processor and the L1 caches and MMU that, together with the core, comprise the core complex.
- Chapter 6, "e600 Core Registers and Instruction Set Summary," provides a listing and description of the e600 registers in reference form.

Part III, “Memory, Peripherals, and I/O Interfaces,” defines the memory and I/O interfaces of the MPC8610 and how these blocks interact with one another and with other blocks on the device. The following chapters are included:

- Chapter 7, “MPX Coherency Module (MCM) Overview,” defines the e600 coherency module and how it facilitates communication between the e600 core complex and the other blocks that comprise the coherent memory domain of the MPC8610.
- Chapter 8, “DDR Memory Controller,” describes the DDR SDRAM memory controller of the MPC8610.
- Chapter 9, “Enhanced Local Bus Controller,” describes the enhanced local bus controller of the MPC8610. The enhanced local bus controller (eLBC) provides a seamless interface to many types of memory devices and peripherals.
- Chapter 10, “Display Interface Unit (DIU),” describes the integrated LCD controller of the MPC8610.
- Chapter 11, “Programmable Interrupt Controller (PIC),” describes the OpenPIC-compliant embedded programmable interrupt controller (PIC) of the MPC8610.
- Chapter 12, “I^2C Modules,” describes the inter-IC (I^2C) bus controller of the MPC8610.
- Chapter 13, “DUART,” describes the (dual) universal asynchronous receiver/transmitters (UARTs) which feature a PC16552D-compatible programming model.
- Chapter 14, “Fast/Serial Infrared Interfaces (FIRI/SIRI),” describes the IrDa-compatible infrared interface controller of the MPC8610.
- Chapter 15, “Serial Peripheral Interface,” describes the serial peripheral interface (SPI) controller of the MPC8610.
- Chapter 16, “Synchronous Serial Interface (SSI),” describes the synchronous serial interface (SSI) controllers of the MPC8610 that implement the inter-IC sound bus standard (I^2S) and Intel AC97 standards for audio.
- Chapter 17, “Global Timer Module,” describes the global timer modules of the MPC8610.
- Chapter 18, “Watchdog Timer,” describes the watchdog timer of the MPC8610.
- Chapter 19, “DMA Controllers,” describes the dual, four-channel general-purpose DMA controllers of the MPC8610.
- Chapter 20, “PCI Bus Interface,” describes the PCI controller and provides a basic description of the PCI bus operations.
- Chapter 21, “PCI Express Interface Controller,” describes the PCI Express® controllers of this device.
- Chapter 22, “General Purpose I/O (GPIO),” describes the two general purpose I/O (GPIO) modules of the MPC8610.

Part IV, “Global Functions and Debug,” defines other global blocks of the MPC8610. The following chapters are included:

- Chapter 23, “Global Utilities,” defines the global utilities of the MPC8610. These include I/O device enabling, power-on-reset (POR) configuration monitoring, and multiplexing controls for the signals.

- Chapter 24, “Device Performance Monitor,” describes the performance monitor of the MPC8610. Note that the MPC8610 performance monitor is similar to but separate from the performance monitor implemented on the e600 core.
- Chapter 25, “Debug Features and Watchpoint Facilities,” describes the debug features, watchpoint monitor, and trace buffers of the MPC8610.

This reference manual also includes the following:

- Appendix A, “Complete List of Configuration, Control, and Status Registers,” provides a complete listing of registers within the CCSR space.
- Appendix B, “Revision History,” lists the major differences between revisions of the MPC8610 *Integrated Host Processor Reference Manual*.
- 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 ISA and computer architecture in general:

- *The PowerPC Architecture: A Specification for a New Family of RISC Processors*, Second Edition, by International Business Machines, Inc.

  For updates to the specification, see
  http://www-1.ibm.com/support/docview.wss?uid=pub1sa14208300
- *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:

- 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 parts, an addendum is 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.

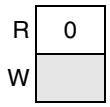
- Product briefs—Each device has a product brief that provides an overview of its suggested applications, critical performance parameters, features, and developer environment.
- Application notes—These short documents address specific design issues useful to programmers and engineers working with Freescale processors.
- *PowerPC e600 Core Complex Reference Manual* (e600CORERM)
- *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture* (MPCFPE32B)

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

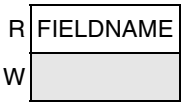
Conventions

This document uses the following notational conventions:

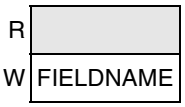
cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
mnemonics	Instruction mnemonics are shown in lowercase bold.
italics	Italics indicate variable command parameters, for example, **bcctr***x*. Book titles in text are set in italics Internal signals are set in lowercase italics, for example, $\overline{core\_int}$
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 an e600 register. Although these bits can be written to as ones or zeros, they are always read as zeros.



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.



Indicates a read-only bit field in a memory-mapped register.



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

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 3.1, "Signals Overview."

Acronyms and Abbreviations

Table i contains acronyms and abbreviations used in this document.

Table i. Acronyms and Abbreviated Terms

Term	Meaning
ADB	Allowable disconnect boundary
ATM	Asynchronous transfer mode
ATMU	Address translation and mapping unit
BD	Buffer descriptor
BIST	Built-in self test
BRI	Basic rate interface
BTB	Branch target buffer
BUID	Bus unit ID
CAM	Content-addressable memory
CCB	Core complex bus
CCSR	Configuration control and status register
CEPT	Conférence des administrations européenes des postes et télécommunications (European Conference of Postal and Telecommunications Administrations)
COL	Collision
CRC	Cyclic redundancy check
CRS	Carrier sense
DDR	Double data rate

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
DMA	Direct memory access
DPLL	Digital phase-locked loop
DRAM	Dynamic random access memory
DUART	Dual universal asynchronous receiver/transmitter
EA	Effective address
ECC	Error checking and correction
ECM	e600 coherency module
EEST	Enhanced Ethernet serial transceiver
EHPI	Enhanced host port interface
EPROM	Erasable programmable read-only memory
FCS	Frame-check sequence
FEC	10/100 fast Ethernet controller
GCI	General circuit interface
GMII	Gigabit media independent interface
GPCM	General-purpose chip-select machine
GPIO	General-purpose I/O
GPR	General-purpose register
GUI	Graphical user interface
HDLC	High-level data link control
I^2C	Inter-integrated circuit
IDL	Inter-chip digital link
IEEE	Institute of Electrical and Electronics Engineers
IPG	Interpacket gap
IrDA	Infrared Data Association
ISDN	Integrated services digital network
ITLB	Instruction translation lookaside buffer
IU	Integer unit
JTAG	Joint Test Action Group
LAE	Local access error
LAW	Local access window
LBC	Local bus controller
LIFO	Last-in-first-out
LRU	Least recently used

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
LSB	Least-significant byte
lsb	Least-significant bit
LSU	Load/store unit
MAC	Multiply accumulate, media access control
MDI	Medium-dependent interface
MESI	Modified/exclusive/shared/invalid—cache coherency protocol
MII	Media independent interface
MMU	Memory management unit
MSB	Most-significant byte
msb	Most-significant bit
NMSI	Nonmultiplexed serial interface
No-op	No operation
OCeaN	On-chip network
OSI	Open systems interconnection
PCI	Peripheral component interconnect
PCMCIA	Personal Computer Memory Card International Association
PCS	Physical coding sublayer
PIC	Programmable interrupt controller
PMA	Physical medium attachment
PMD	Physical medium dependent
POR	Power-on reset
PRI	Primary rate interface
RGMII	Reduced gigabit media independent interface
RISC	Reduced instruction set computing
RTOS	Real-time operating system
RWITM	Read with intent to modify
RWM	Read modify write
Rx	Receive
RxBD	Receive buffer descriptor
SCC	Serial communication controller
SCP	Serial control port
SDLC	Synchronous data link control
SDMA	Serial DMA

Table i. Acronyms and Abbreviated Terms (continued)

Term	Meaning
SFD	Start frame delimiter
SI	Serial interface
SIU	System interface unit
SMC	Serial management controller
SNA	Systems network architecture
SPI	Serial peripheral interface
SPR	Special-purpose register
SRAM	Static random access memory
TAP	Test access port
TBI	Ten-bit interface
TDM	Time-division multiplexed
TLB	Translation lookaside buffer
TSA	Time-slot assigner
TSEC	Three-speed Ethernet controller
Tx	Transmit
TxBD	Transmit buffer descriptor
UART	Universal asynchronous receiver/transmitter
UPM	User-programmable machine
USB	Universal serial bus
UTP	Unshielded twisted pair
VA	Virtual address
ZBT	Zero bus turnaround

Diagrams

The diagrams in this publication are provided to aid in understanding the overall functionality and features of this product. They do not depict the implementation details of the product, which are subject to change.

Part I
Overview

Part I describes the many features of the MPC8610 integrated host processor at an overview level. The following chapters are included:

- Chapter 1, “MPC8610 Overview,” provides a high-level description of features and functionality of the MPC8610 integrated processor. It describes the MPC8610, its interfaces, and programming model. The functional operation of the MPC8610, with emphasis on peripheral functions, is also described.
- Chapter 2, “Memory Map,” describes the mechanisms that define the device memory map—the local access windows (LAWs), the address translation and mapping units (ATMUs), and the configuration, control, and status registers (CCSRs).
- Chapter 3, “Signal Descriptions,” provides a listing of all the external signals, cross-references for signals that serve multiple functions, output signal states at reset, and reset configuration signals (and the modes they define).
- Chapter 4, “Reset, Clocking, and Initialization,” describes the hard and soft resets, power-on reset sequence (POR), power-on reset configuration, clocking, and initialization of the MPC8610.

Chapter 1
MPC8610 Overview

This chapter provides a high-level description of the features and functionality of the MPC8610 integrated host processor.

The MPC8610 is a member of a growing family of products that combine system-level support for industry standard interfaces to processor cores built on Power Architecture™ technology. The MPC8610 integrates an e600 core with on-chip controllers and interfaces to support image processing as well as general purpose embedded applications offering high performance with low power consumption, and low system cost.

The e600 core is a high-performance, superscalar design supporting multiple execution units, including four independent units that execute the AltiVec® instruction set architectural extension for support of 128-bit vector operations. The e600 core features separate 32-Kbyte, level-one (L1) instruction and data caches, and a 256-Kbyte L2 cache. Because the e600 core is the essentially the same processor as that used in Freescale's MPC744x and MPC864x families, software applications written for those devices should port easily to the MPC8610.

The high level of integration in the MPC8610 helps simplify board design and offers significant bandwidth and performance increase. The MPC8610 has a DDR/DDR2 SDRAM memory controller, an enhanced local bus controller (eLBC), a display interface unit (LCD controller), a programmable interrupt controller (PIC), two I^2C controllers, two dual universal asynchronous receiver/transmitters (DUARTs) for serial communication, two Fast Infra-Red interfaces (FIRI), a serial peripheral interface (SPI), two synchronous serial interface (SSI) controllers for I^2S or AC97 audio, two global timer modules, a watchdog timer facility, two four-channel DMA controllers, and a 32-bit, 66-MHz PCI interface. For high-speed interconnect, the MPC8610 provides a 1x/2x/4x PCI Express® interface and a 1x/2x/4x/8x PCI Express interface.

1.1 MPC8610 Overview

This section provides a high-level overview of the MPC8610 features. Figure 1-1 shows the major functional units within the MPC8610.

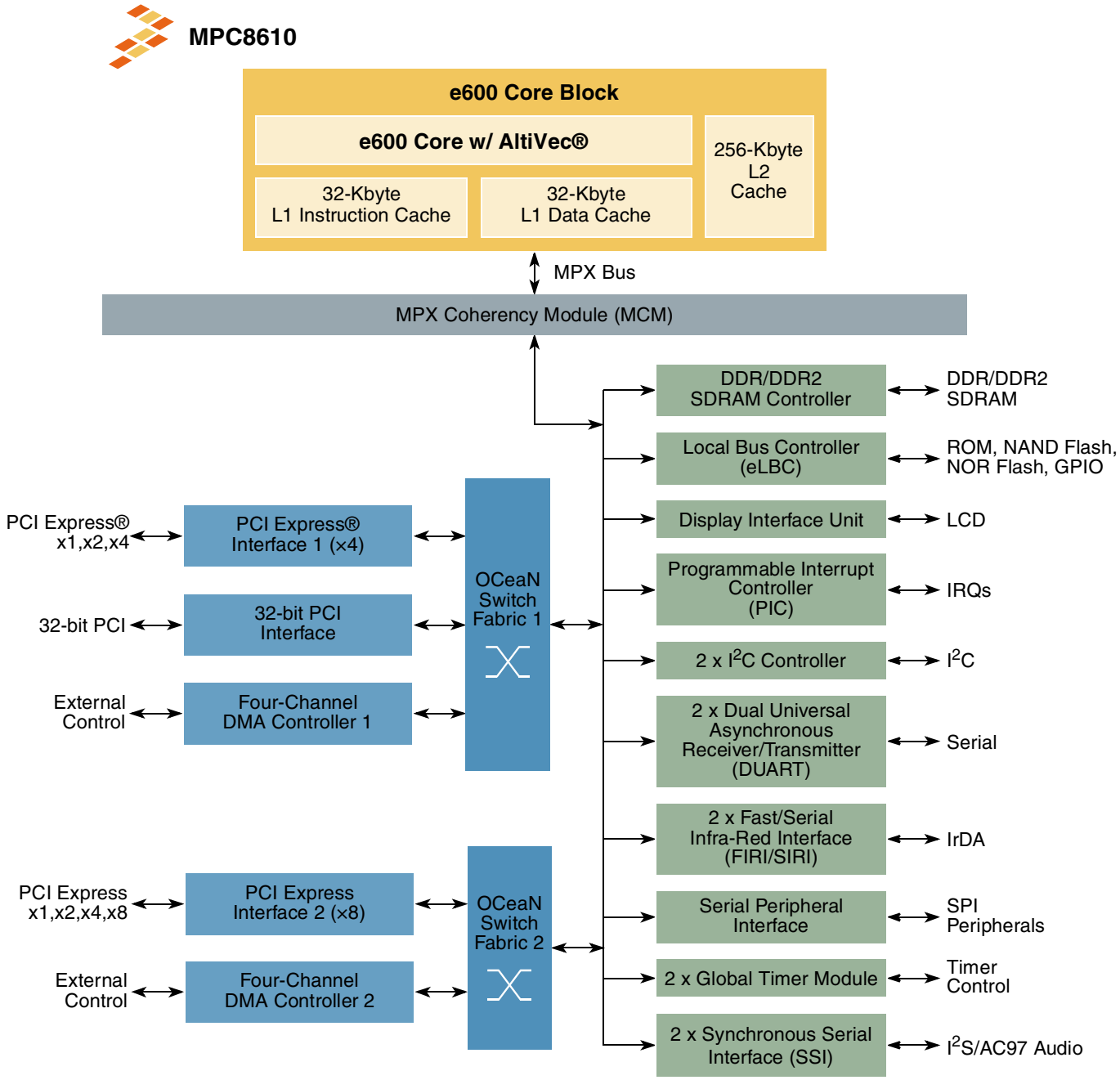


Figure 1-1. MPC8610 Block Diagram

1.1.1 Key Features

The key features of the MPC8610 are as follows:

- e600 processor core
  - High-performance, 32-bit superscalar processor that implements Power Architecture technology
  - Eleven independent execution units and three register files
    - Branch processing unit (BPU)
    - Four integer units (IUs) that share 32 GPRs for integer operands
    - 64-bit floating-point unit (FPU)
    - Four vector units and a 32-entry vector register file (VRs)
    - Three-stage load/store unit (LSU)
  - Three issue queues, FIQ, VIQ, and GIQ, can accept as many as one, two, and three instructions, respectively, in a cycle.
  - Rename buffers
  - Dispatch unit
  - Completion unit
  - Two separate 32-Kbyte instruction and data L1 caches
  - Integrated 256-Kbyte, eight-way set-associative unified instruction and data L2 cache with ECC
  - 36-bit real addressing
  - Separate memory management units (MMUs) for instructions and data
  - Power and thermal management
  - Performance monitor
  - In-system testability and debugging features
  - Reliability and serviceability
- MPX coherency module (MCM)
  - Provides coherency management for the SoC
- Flexible memory map
  - Eight local access windows define mapping within local 36-bit address space
  - Inbound and outbound address translation and mapping units (ATMUs) map to larger external address spaces
    - Three inbound windows plus a configuration window for PCI and each PCI Express interface
    - Four outbound windows plus default translation for PCI and each PCI Express interface
- DDR memory controller
  - 64- or 32-bit data path (72-bit with ECC)
  - Programmable timing support for DDR and DDR2 SDRAM
  - Up to 533-MHz DDR2 data rate and up to 400-MHz DDR data rate

  - Error checking and correction (ECC) using eight additional bits
  - Single-bit error correction/double-bit error detection within a nibble
  - SDRAM chip configurations from 64 Mbit to 4 Gbit organized as x8, x16, and x32 (x4 not supported)
  - Four chip selects
  - Up to 16 Gbytes memory
  - 4 or 8 sub-banks per chip select
  - Page mode support
  - Up to 32 simultaneous open pages for DDR2; Up to 16 simultaneous open pages for DDR
  - Contiguous or discontiguous memory mapping
  - On-die termination support when using DDR2
  - On-the-fly power management using CKE signal
  - Sleep mode support for self-refresh SDRAM
  - Supports auto-refresh
  - Registered DIMM support
  - 1.8 V SSTL_1.8 compatible I/O for DDR2; 2.5 V SSTL_2 compatible I/O for DDR
  - Error injection for diagnostic purposes
- Enhanced local bus controller (eLBC)
  - Multiplexed 32-bit address and data operating at up to 133 MHz
  - The 32-, 16-, and 8-bit port sizes
  - Eight chip-selects support eight external targets
  - Three protocol engines available on a per chip select basis:
    - General-purpose chip select machine (GPCM) compatible with SRAM, EPROM, NOR flash E^2PROM, and peripherals
    - NAND flash control machine (FCM) compatible with NAND flash E^2PROM
    - Three user programmable machines (UPMs) compatible with DRAMs and burstable SRAMs
  - Up to eight-beat burst transfers
  - Default boot ROM chip select with configurable bus width (8-,16-, or 32-bit)
  - Six general purpose output pins
  - Variable memory block sizes (32 Kbytes to 4 Gbytes)
  - Odd/even parity checking including read-modify-write (RMW) parity for single accesses
  - Parity byte-select
  - Write-protection capability
  - Atomic operation
  - 3.3 V, 2.5 V, or 1.8 V I/O
- Display Interface Unit (DIU)
  - Supports 12/16/18/24-bit color TFT panels and 8-bit monochrome LCD panels

  - Supports any resolution up to 1280×1024 (single-plane), or any resolution up to 1024×768 (3 multiple planes)
  - Display refresh rate: 60 Hz (up to 1280×1024) or 72 Hz (up to 1024×768)
  - Display color depth: up to 24 bpp
  - Display Interface: Parallel TTL (24 data signals, Clock, Hsync, Vsync, and Enable) compatible with TTL-to-LVDS devices
- Programmable interrupt controller (PIC)
  - Programming model is compliant with the OpenPIC architecture
  - Supports 16 programmable interrupt and processor task priority levels
  - Supports 12 discrete external interrupts and 48 internal interrupts
  - Eight global high resolution timers/counters that can generate interrupts
  - Support for PCI-Express message shared interrupts (MSIs)
- Two I^2C controllers
  - Each controller features a two-wire interface
  - Multiple master support
  - Master or slave I^2C mode support
  - On-chip digital filtering rejects spikes on the bus
  - Boot sequencer
    - Optionally loads configuration data from serial ROM at reset via the I^2C interface
    - Can be used to initialize configuration registers and/or memory
    - Supports extended I^2C addressing mode
    - Data integrity checked with preamble signature and CRC
- Two DUART modules
  - Programming model compatible with the 16450 UART and the PC16550D
  - Flexible pinout multiplexing; can be configured as:
    - Four 2-wire interfaces (SIN, SOUT)
    - Two 4-wire interfaces (SIN, SOUT, $\overline{\text{RTS}}$, $\overline{\text{CTS}}$)
    - One 4-wire and one 2-wire interface
- Two Infrared interfaces
  - Used for non-wired communications
  - Two 2-wire ports
  - IrDA, version 1.4 compatible
  - Each port supports the following physical layer protocols:
    - 4 Mbps fast infrared (FIR)
    - 0.576 Mbps and 1.152 Mbps medium infrared (MIR)
    - 9.6 kbps to 115.2 kbps serial infrared (SIR)
- Serial peripheral interface (SPI)
  - Master or slave support

- Two synchronous serial interface (SSI) modules
  - Inter-IC sound (I^2S) bus or AC97 audio
- Two global timer modules (GTMs)
- Watchdog timer (WDT)
  - Asserts external HRESET_REQ signal if the WDT times out
  - Software periodically resets the WDT within timeout period
  - 8 sec default timeout period @ 533 MHz platform frequency
  - Software can shorten timeout period only once during initialization (write-once); software cannot increase the timeout period
  - POR configuration signal to enable or disable WDT
- Two integrated DMA controllers
  - Each controller has four channels
  - All channels accessible by both the local and the remote masters
  - Supports transfers to or from any local memory or I/O port
  - Ability to start and flow control each DMA channel from external 3-pin interface
  - Each controller has a 1-Kbyte read buffer
- PCI interface
  - Complies with *PCI Local Bus Specification, Revision 2.2*
  - 32-bit interface
  - Supports 33 or 66 MHz bus frequency
  - 3.3 V I/O
  - Address translation and mapping units (ATMUs)
    - Three inbound windows plus a configuration window
    - Four outbound windows plus default translation
- Two PCI Express interfaces
  - Compatible with *PCI Express Base Specification, Revision 1.0a*
  - Both controllers may be configured (at reset) as a root complex (RC) or endpoint (EP) device
  - PCI Express controller 1 supports ×1, ×2, and ×4 link widths;
    PCI Express controller 2 supports ×1, ×2, ×4, and ×8 link widths
  - 2.5 Gbaud (signalling), 2.0 Gbps (data rate) per lane per direction
  - One virtual channel (VC0) and one traffic class (TC0)
  - 256-byte maximum payload size
  - Address translation and mapping units (ATMUs)
    - Three inbound windows plus a configuration window on each PCI Express block
    - Four outbound windows plus default translation for each PCI Express block
- GPIO
  - 2 GPIO Modules—GPIO1 and GPIO2
  - GPIO1 has 32 signals (16 input/output and16 output-only)

  - GPIO2 has 27 signals (26 input/output and 1 input-only)
  - 3.3 V
- Device power management
  - Supports nap and sleep modes
  - Configuration option to withhold clocks from select blocks (eLBC, DDR, PCI, PCI Express 1 & 2, DUART 1 & 2, DIU, SPI, I^2C, FIRI/SIRI 1 & 2, GTM 1 & 2, and SSI 1 & 2) to reduce power
- Thermal diode support
- Device performance monitor
  - Supports eight 32-bit counters that count the occurrence of selected events
  - Ability to count up to 512 counter-specific events
  - Supports 64 reference events that can be counted on any of the 8 counters
  - Supports duration and quantity threshold counting
  - Burstiness feature that permits counting of burst events with a programmable time between bursts
  - Triggering and chaining capability
  - Ability to generate an interrupt on overflow
- Debug features
  - 64 bit × 256 entry trace buffer
  - Watchpoint monitor
    - Two-level triggering
    - External TRIG_OUT signal
- IEEE 1149.1-compliant, JTAG boundary scan
  - Also supports common on-chip processor (COP) for third-party debuggers

1.2 MPC8610 Architecture Overview

The following sections describe the major functional units of the MPC8610.

1.2.1 e600 Core

Key features of the e600 core are as follows:

- High-performance, superscalar microprocessor
  - As many as 4 instructions can be fetched from the instruction cache at a time
  - As many as 3 instructions can be dispatched to the issue queues at a time
  - As many as 12 instructions can be in the instruction queue (IQ)
  - As many as 16 instructions can be at some stage of execution simultaneously
  - Single-cycle execution for most instructions
  - One instruction per clock cycle throughput for most instructions

— Seven-stage pipeline control

- Eleven independent execution units and three register files
  — Branch processing unit (BPU) features static and dynamic branch prediction
    – 128-entry (32-set, four-way set-associative) branch target instruction cache (BTIC), a cache of branch instructions that have been encountered in branch/loop code sequences. If a target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, a fetch that hits the BTIC provides the first four instructions in the target stream.
    – 2048-entry branch history table (BHT) with two bits per entry for four levels of prediction—not-taken, strongly not-taken, strongly taken
    – Up to three outstanding speculative branches
    – Branch instructions that do not update the count register (CTR) or link register (LR) are often removed from the instruction stream.
    – 8-entry link register stack to predict the target address of Branch Conditional to Link Register (**bclr**) instructions.
  — Four integer units (IUs) that share 32 GPRs for integer operands
    – Three identical low latency IUs (IU1a, IU1b, and IU1c) can execute all integer instructions except multiply, divide, and move to/from special-purpose register instructions.
    – IU2 executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions.
  — 64-bit floating-point unit (FPU)
    – Five-stage FPU
    – Fully IEEE 754-1985-compliant FPU for both single- and double-precision operations
    – Supports non-IEEE mode for time-critical operations
    – Hardware support for denormalized numbers
    – Thirty-two 64-bit FPRs for single- or double-precision operands
  — Four vector units and 32-entry vector register file (VRs) implementing the AltiVec instruction set
    – Vector permute unit (VPU)
    – Vector integer unit 1 (VIU1) handles short-latency AltiVec integer instructions, such as vector add instructions (**vaddsbs**, **vaddshs**, and **vaddsws**, for example)
    – Vector integer unit 2 (VIU2) handles longer-latency AltiVec integer instructions, such as vector multiply add instructions (**vmhaddshs**, **vmhraddshs**, and **vmladduhm**, for example).
    – Vector floating-point unit (VFPU)
  — Three-stage load/store unit (LSU)
    – Supports integer, floating-point and vector instruction load/store traffic
    – Four-entry vector touch queue (VTQ) supports all four architected AltiVec data stream operations

  - Three-cycle GPR and AltiVec load latency (byte, half-word, word, vector) with 1 cycle throughput
  - Four-cycle FPR load latency (single, double) with 1 cycle throughput
  - No additional delay for misaligned access within double-word boundary
  - Dedicated adder calculates effective addresses (EAs)
  - Supports store gathering
  - Performs alignment, normalization, and precision conversion for floating-point data
  - Executes cache control and translation lookaside buffer (TLB) instructions
  - Performs alignment, zero padding, and sign extension for integer data
  - Supports hits under misses (multiple outstanding misses)
  - Supports both big- and little-endian modes, including misaligned little-endian accesses
- Three issue queues FIQ, VIQ, and GIQ can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  - Instructions can be dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  - A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  - Space must be available in the CQ for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not in an issue queue).
- Rename buffers
  - 16 GPR rename buffers
  - 16 FPR rename buffers
  - 16 VR rename buffers
- Dispatch unit
  - The decode/dispatch stage fully decodes each instruction.
- Completion unit
  - The completion unit retires an instruction from the 16-entry completion queue (CQ) when all instructions ahead of it have been completed, the instruction has finished execution, and no exceptions are pending.
  - Guarantees sequential programming model (precise exception model)
  - Monitors all dispatched instructions and retires them in order
  - Tracks unresolved branches and flushes instructions after a mispredicted branch
  - Retires as many as three instructions per clock cycle
- L1 cache had the following characteristics:
  - Two separate 32-Kbyte instruction and data caches (Harvard architecture).
  - Instruction and data caches are eight-way set-associative.
  - Instruction and data caches have 32-byte cache blocks. A cache block is the block of memory that a coherency state describes—corresponds to a cache line for the L1 data cache.
  - Cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.

- The caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way.
- Cache write-back or write-through operation programmable on a per-page or per-block basis
- Instruction cache can provide four instructions per clock cycle; data cache can provide four words per clock cycle
  - Two-cycle latency and single-cycle throughput for instruction or data cache accesses.
- Caches can be disabled in software
- Caches can be locked in software
- Supports a four-state modified/exclusive/shared/invalid (MESI) coherency protocol.
  - A single coherency status bit for each instruction cache block allows encoding for the following two possible states:

    Invalid (INV)

    Valid (VAL)
  - Two status bits (MESI[0–1]) for each data cache block allow encoding for coherency, as follows:

    00 = invalid (I)

    01 = shared (S)

    10 = exclusive (E)

    11 = modified (M)
- Separate copy of data cache tags for efficient snooping
- Both the L1 caches support parity generation and checking (enabled through bits in the ICTRL register) as follows:
  - Instruction cache—one parity bit per instruction
  - Data cache—one parity bit per byte of data
- No snoopings of instruction cache except for **icbi** instruction
- Data cache supports AltiVec LRU and transient instructions, as described in Section 1.2.2.2, “AltiVec Instruction Set.”
- Critical double- and/or quad-word forwarding is performed as needed. Critical quad-word forwarding is used for AltiVec loads and instruction fetches. Other accesses use critical double-word forwarding.

• On-chip Level 2 (L2) cache has the following features:
- Integrated 256-KByte, eight-way set-associative unified instruction and data cache for the e600 core.
- Fully pipelined to provide 32 bytes per clock cycle to the L1 caches.
- Total latency of 11.5 processor cycles for L1 data cache miss that hits in the L2 if ECC is disabled. When ECC is enabled, the L2 load access time is 12.5 cycles. The L2 cache is fully pipelined for a 2-cycle throughput.
- Uses one of two random replacement algorithms (selectable through L2CR).
- Cache write-back or write-through operation programmable on a per-page or per-block basis

  - Organized as 32 bytes/block and 2 blocks (sectors)/line (a cache block is the block of memory that a coherency state describes).
  - Supports error correction code (ECC) generation and checking for both tags and data (enabled through L2CR).
- Separate memory management units (MMUs) for instructions and data
  - 52-bit virtual address; 32- or 36-bit physical address
  - Address translation for 4-Kbyte pages, variable-sized blocks, and 256-Mbyte segments
  - Memory programmable as write-back/write-through, caching-inhibited/caching-allowed, and memory coherency enforced/memory coherency not enforced on a page or block basis
  - Separate IBATs and DBATs (eight each) also defined as SPRs
  - Separate instruction and data translation lookaside buffers (TLBs)
    - Both TLBs are 128-entry, two-way set-associative, and use LRU replacement algorithm
    - TLBs are hardware- or software-reloadable (that is, on a TLB miss, a page table search is performed in hardware or by system software)
- Efficient data flow
  - L1/L2 bus interface allows up to 256 bits.
  - The L1 data cache is fully pipelined to provide 128 bits/cycle to or from the VRs.
  - L2 cache is fully pipelined to provide 256 bits per processor clock cycle to the L1 cache.
  - As many as eight outstanding, out-of-order cache misses are allowed between the L1 data cache and L2 bus.
  - As many as 16 out-of-order transactions can be present on the MPX bus.
  - Store merging for multiple store misses to the same line. Only coherency action taken (address-only) for store misses merged to all 32 bytes of a cache block (no data tenure needed).
  - Three-entry finished store queue and five-entry completed store queue between the LSU and the L1 data cache
  - Separate additional queues for efficient buffering of outbound data (such as castouts and write-through stores) from the L1 data cache and L2 cache
- Multiprocessing support features include the following:
  - Hardware-enforced, MESI cache coherency protocols for data cache
  - Load/store with reservation instruction pair for atomic memory references, semaphores, and other multiprocessor operations
- Power and thermal management
  - Employs dynamic power management, which automatically minimizes power consumption of blocks when they are idle
  - Three programmable power states are available to the system: full power, doze, nap, and sleep.
  - Instruction cache throttling provides control of instruction fetching to limit device temperature.
  - Temperature Diode
    - Can be used in conjunction with external devices to monitor junction temperature
- Performance monitor can be used to help debug system designs and improve software efficiency.

- In-system testability and debugging features through JTAG boundary-scan capability
- Reliability and serviceability
  - Parity checking on the MPX bus

The e600 is a superscalar processor that can dispatch and complete three instructions simultaneously. The ability to execute several instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for e600 core-based systems. Most integer instructions (including VIU1 instructions) have a 1-clock cycle execution latency.

Several execution units feature multiple-stage pipelines; that is, the tasks they perform are broken into subtasks executed in successive stages. Typically, instructions follow one another through the stages, so a four-stage unit can work on four instructions when its pipeline is full. So, although an instruction may have to pass through several stages, the execution unit can achieve a throughput of one instruction per clock cycle.

AltiVec computational instructions are executed in the four independent, pipelined AltiVec execution units. A maximum of two AltiVec instructions can be issued in order to any combination of AltiVec execution units per clock cycle. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions. The VPU has a two-stage pipeline; the VIU2 and VFPU each have four-stage pipelines. As many as 10 AltiVec instructions can be executing concurrently.

Note that for the e600 core, double- and single-precision versions of floating-point instructions have the same latency. For example, a floating-point multiply-add instruction takes five cycles to execute, regardless of whether it is single- (**fmadds**) or double-precision (**fmadd**).

The e600 core has independent on-chip, 32-Kbyte, eight-way set-associative, physically addressed L1 caches for instructions and data. The e600 core’s L2 cache is implemented with an on-chip, 256-Kbyte, eight-way set-associative physically addressed memory available for storing data, instructions, or both. The L2 cache supports parity and ECC generation and checking for both tags and data. The L2 cache is fully pipelined for single-cycle throughput.

The e600 core has independent instruction and data memory management units (MMUs). Each MMU has a 128-entry, two-way set-associative translation lookaside buffer (DTLB and ITLB) that saves recently used page address translations. Block address translation is implemented with the eight-entry instruction and data block address translation (IBAT and DBAT) arrays defined by the PowerPC Architecture. During block translation, effective addresses are compared simultaneously with all BAT entries.

1.2.2 MPX Coherency Module (MCM)

The MPX coherency module (MCM) provides coherency management for the SoC. The MCM is responsible for tracking the coherent state of transactions that are part of the coherent domain. It handles any necessary reordering to resolve all coherent memory references. Once addresses are snooped clean they are sent on to the memory controller. Once a transaction has been accepted by the MCM it is considered globally visible. The MCM also arbitrates competing requests for both the MPX bus and the platform bus.

1.2.3 Address Map

The MPC8610 supports a flexible 36-bit physical address map. Conceptually, the address map consists of local space and external address space. The local address map is defined by a set of ten local access windows (LAWs). Each of these windows maps a region of the local address space to a specified target interface, such as the DDR controller, enhanced local bus controller, PCI Express controllers, or other targets. The internal configuration, control, and status registers (CCSRs) for all the functional blocks are located in the local memory space at a specific CCSR window.

If the target mapping performed by the local access windows directs the transaction to one of the external interfaces (as an outbound read or write), the transaction is then mapped into that interface's external address space by address translation and mapping unit (ATMU) windows associated with the external interface. Outbound ATMUs perform the mapping from the local 36-bit address space to the address space of the external interface; inbound ATMU windows perform the address translation from the external address space to the local address space.

1.2.4 DDR/DDR2 SDRAM Controller

The MPC8610 supports a variety of SDRAM configurations which can be configured to support the various memory sizes through software initialization of on-chip configuration registers. SDRAM banks can be built using DIMMs or directly-attached memory devices. Sixteen multiplexed address signals provide for device densities of 64 Mbit to 4 Gbit. Four chip select signals support up to four banks of memory. The MPC8610 supports bank sizes from 64 Mbytes to 4 Gbytes. The theoretical maximum amount of memory supported is achieved using four banks of ×8, 4-Gbit devices, allowing the memory controller to address up to 16 Gbytes. Nine-column address strobes (MDM[0:8]) are used to provide byte selection for memory bank writes.

The MPC8610 can be configured to retain the currently active SDRAM page for pipelined burst accesses. Page mode support of up to 32 simultaneously open pages can dramatically reduce access latencies for page hits. Depending on the memory system design and timing parameters, using page mode can save 3 to 4 clock cycles from subsequent burst accesses that hit in an active page.

The MPC8610 can invoke a level of system power management by asserting the CKE SDRAM signal on-the-fly to put the memory into a low-power sleep mode.

1.2.5 Enhanced Local Bus Controller (eLBC)

The enhanced local bus controller (eLBC) allows connections with a wide variety of external memories, DSPs, and ASICs.

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.

Three separate state machines share the same external pins and can be programmed separately to access different types of devices.

- 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 E^2PROMs with 8-bit and 16-bit data buses. The FCM has an automatic boot-loading feature that allows the CPU to boot from high density E^2PROM, 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 permits 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 FCM permits zero-overhead error checking and correction in both boot blocks and page data transfers.
- 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.

1.2.6 Display Interface Unit (DIU)

The DIU is a display controller designed to handle TFT LCD displays. The DIU does not have internal frame buffers; it reads the data from main memory at the same rate at which it refreshes the display. Besides generating all the signals required to drive the display, the DIU handles the real-time blending of up to three planes onto the display. Alpha blending is performed between the planes. Chroma key support is also present to help relieve the host processor from all the very computational and bandwidth-consuming blending tasks while simultaneously allowing the users to maintain the graphics quality required by many applications.

1.2.7 Programmable Interrupt Controller (PIC)

- OpenPIC compatible programming model
- Support for 12 external and 48 internal interrupt sources
- 16 programmable interrupt priority levels
- Four 32-bit messaging interrupt channels for a total of 256 interrupts that are used with message signalled interrupts (MSIs)

- Fully-nested interrupt delivery
- Processor initialization control
- Programmable resetting of the PIC unit through the global configuration register
- Interrupts can be routed to external pin for connection of external interrupt controller device such as an 8259 programmable interrupt controller
- In 8259 mode, it generates a local (internal) interrupt output signal, IRQ_OUT
- Recovery from spurious interrupts
- Four global high resolution timers/counters that can generate interrupts
- Interrupts can be routed to the e600 core's standard or critical interrupt inputs.
- Interrupt summary registers allow fast identification of interrupt source.

The programmable interrupt controller (PIC) supports the e600 core and implements the necessary functions to provide a flexible solution for a general-purpose interrupt control. The interrupt controller unit implements the logic and programming structures of the OpenPIC architecture.

1.2.8 I^2C Controllers

The MPC8610 supports two I^2C controllers. The inter-IC (IIC or I^2C) bus is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange between devices. The synchronous, multi-master bus of the I^2C allows the MPC8610 to exchange data with other I^2C devices such as microcontrollers, EEPROMs, real-time clock devices, A/D converters, and LCDs. The two-wire bus (serial data SDA and serial clock SCL) minimizes the interconnections between devices.

The I^2C controller is a true multimaster 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^2C controller consists of a transmitter/receiver unit, a clocking unit, and a control unit. The dual I^2C units supports general broadcast mode and on-chip filtering rejects spikes on the bus.

1.2.9 Boot Sequencer

The MPC8610 provides a boot sequencer that uses the I^2C interface to access an external serial ROM and loads the data into the MPC8610's configuration registers. The boot sequencer is enabled by a configuration pin sampled at the negation of the MPC8610 hardware reset signal. If enabled, the boot sequencer holds the MPC8610 e600 processor core in reset until the boot sequence is complete. If the boot sequencer is not enabled, the e600 processor core exits reset and fetches boot code in default configurations.

1.2.10 Dual Universal Asynchronous Receiver/Transmitter (DUART)

The MPC8610 includes two DUARTs intended for use in maintenance, bringing-up, and debugging of systems. The MPC8610 provides up to four 2-wire (SIN*n* and SOUT*n*) or two 4-wire (SIN*n*, SOUT*n*, $\overline{\text{RTS}}$*n*, $\overline{\text{CTS}}$*n*) ports on the available signals. 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, line status, and MODEM status.

The MPC8610 DUART supports a full-duplex operation. It is compatible with the PC16450 and PC16550 programming models. Also, 16-byte FIFOs are supported for both the transmitter and the receiver.

Software programmable baud 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.

1.2.11 Fast Infra-Red Interfaces

The IrDA-compliant fast/serial infrared controllers support packet-based high-,medium-, and low-speed infrared communications using an external transceiver device. These controllers have two sets of resources, serial infrared (SIRI) and fast infrared (FIRI), that support the following speeds:

- High speed—4 Mbits/s (FIRI)
- Medium speed—0.576 and 1.152 Mbits/s (FIRI)
- Low speed—9.6–115.2 Kbits/s (SIRI)

Communication using the infrared interfaces is usually handled with DMA requests, in which the DMA transfers data to and from FIFOs within the modules; each module has trigger points to generate a DMA request when the FIFO is approaching full (receive) or becoming empty (transmit). The FIRI module has two independent 128-byte FIFOs for transmitter and receiver. The SIRI module has a 32-byte send FIFO and a 32-half-word receive FIFO.

1.2.12 Serial Peripheral Interface (SPI)

The serial peripheral interface (SPI) allows the MPC8610 to exchange data between other devices such as external PHYs for configuration, 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.

1.2.13 Synchronous Serial Interface (SSI)

The Synchronous Serial Interface (SSI) is a full-duplex, serial port that allows the chip to communicate with a variety of serial devices. These serial devices can be standard coder-decoder (CODECs), Digital Signal Processors (DSPs), microprocessors, peripherals, and popular industry audio CODECs that implement the inter-IC sound bus standard (I^2S) and Intel AC97 standard.

1.2.14 DMA Controllers

The MPC8610 supports two DMA engines which are each 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.

DMA transfers can be initiated by either local or remote masters by a single write to a configuration register. There is also support for external control of transfers using $\overline{\text{DMA\_DREQ}}$, $\overline{\text{DMA\_DACK}}$, and $\overline{\text{DMA\_DDONE}}$ handshake signals.

DMA descriptors encompass a rich set of attributes that allow DMA transfers to bypass outbound address translation and supply external addresses and attributes directly. Local attributes such as snoop can be specified by descriptors.

Interrupts are provided on a completed segment, link, list, chain, or on an error condition. Coherency is selectable and hardware enforced (snoop/no snoop).

1.2.15 PCI Interface

The 32-bit PCI controller is compatible with the *PCI Local Bus Specification, Revision 2.2*. The MPC8610 can function as either a PCI host or agent device. The interface supports 32- and 64-bit addressing. As a master, the MPC8610 supports read and write operations to PCI memory space, PCI I/O space, and PCI configuration space. The MPC8610 can also generate PCI special-cycle and interrupt-acknowledge commands. As a target, the MPC8610 supports read and write operations to system memory as well as configuration accesses. An internal arbiter can be used to support up to five external masters using a round-robin arbitration algorithm with two priority levels.

1.2.16 PCI Express Interface

The MPC8610 features two independent PCI Express interface controllers. PCI Express controller 1 supports up to a ×4 link on SerDes port 1 and PCI Express controller 2 supports up to a ×8 link on SerDes port 2. Note that as shown in Figure 1-1, these controllers are on separate on-chip networks and, as such, are non-bridging.

Each PCI Express controller connects the internal platform to a 2.5-GHz serial interface. The PCI Express controller can be configured to operate as either a PCI Express root complex (RC) or an endpoint (EP) device. An RC device connects the host CPU/memory subsystem to I/O devices while an EP device typically denotes a peripheral or I/O device.

Upon coming out of reset, the PCI Express interface performs link width negotiation and exchanges flow control credits with its link partner. Once link autonegotiation is successful, the controller is in operation.

As both an initiator and a target device, the PCI Express interface is capable of high-bandwidth data transfer and is designed to support next generation I/O devices. As an initiator, the PCI Express controller supports memory read and write operations with a maximum transaction size of 256 bytes. In addition, configuration and I/O transactions are supported if the PCI Express controller is in RC mode. As a target interface, the PCI Express controller accepts read and write operations to local memory space. When configured as an EP device, the PCI Express controller accepts configuration transactions to the internal PCI Express configuration registers. Message generation and acceptance are supported in both RC and EP modes. Locked transactions and inbound I/O transactions are not supported.

1.2.17 Power Management

The e600 core is designed for low-power operation. It provides both automatic and program-controlled power reduction modes. If an e600 core’s functional unit is idle, it automatically goes into a low-power mode. This mode does not affect operational performance.

Dynamic power management automatically supplies or withholds power to execution units individually, based upon the contents of the instruction stream. The e600 core execution unit has an independent clock input that is controlled automatically on a clock-by-clock basis; for example, clocking is withheld from the floating-point unit if no floating-point instructions are being executed. The operation of dynamic power management is transparent to software or any external hardware. To save power, the memory controller deasserts the CKE signal to the DRAM when transactions are not being issued to the DRAM. The memory controller can also put DRAM into self-refresh mode when it is in sleep or doze state.

The e600 core provides an instruction cache throttling mechanism to effectively reduce the instruction execution rate without the complexity and overhead of dynamic clock control. When used with dynamic power management, instruction cache throttling provides the system designer with a flexible way to control device temperature while allowing the processor to continue operating.

The instruction cache throttling mechanism simply reduces the instruction dispatch rate. Normally, as many as three instructions are dispatched each cycle. For thermal management, the e600 core provides a supervisor-level instruction cache throttling control register (ICTC) that sets a specific number (1–255) of dead cycles between instructions.

In addition to low-voltage operation and dynamic power management in its execution units, the MPC8610 supports three programmable power consumption modes: full-power, nap, and sleep. These modes can be entered under software control in the e600 core or by external masters accessing a configuration register.

The following four programmable power states are available to the system:

- Full-power—Normal operation. All clocks are on, instructions are fetched and executed.
- Nap—All core clocks are off. Platform clocks remain running. Instruction dispatch is halted.
- Sleep—All core clocks are off. Platform clocks remain running. Instruction dispatch is halted.

Unused blocks can be powered down to reduce power consumption. The configuration register can be written by either the e600 core or an external device. The peripherals that are subject to being powered down in this manner include: the local bus, PCI Express, the memory controller, the DMA controller, the DUART, and the I^2C interfaces.

The MPC8610 has a temperature diode on the microprocessor that can be used in conjunction with other system temperature monitoring devices (such as Analog Devices ADT7461TM). These devices use the negative temperature coefficient of a diode operated at a constant current to determine the temperature of the microprocessor and its environment.

The memory controller has two power management features:

- Dynamic power management mode. The memory controller can deassert the ‘CKE’ pin to the DRAM when there are not any transactions that need to be issued to the DRAM. This will help in power savings.
- Sleep mode. The memory controller can be told to sleep. In this mode, the memory controller will put the DRAM into self-refresh mode (if ‘SREN’ bit is set) as soon as possible. After this, all memory controller clocks may be stopped.

1.2.18 Clocking

The MPC8610 has a system clock input. The e600 core runs at a multiple of this clock. The system logic is synchronous to the system clock, including the DRAM interface. The multipliers define the relationship between the core speed and the DRAM speed. For instance, if a 267 MHz clock rate (533 MHz data rate) DDR2 DRAM is used and a multiplier of 2 is used, then the core will run at 1066 MHz (533 x 2). The L1 and L2 caches run at the same frequency as the core. The lowest available multiple is 2:1, and increments by 0.5 (2.5:1, 3:1, 3.5:1, and continues accordingly) to 14:1.

The internal 128-bit bus runs at the same frequency as the DRAM data rate, as does the internal MPX bus. The local bus is also synchronous to the DRAM speed, with multipliers of 2, 4, 8, and 16 between the local bus speed and the DRAM data rate. Two clock outputs are generated. The OCN switch fabric runs at half speed of the DRAM.

The high-speed interfaces (PCI Express) ports all share two independent PLLs. For each interface, the transmit clock can be sourced from the receive clock, the system clock, or a special differential clock input.

The MPC8610 generates six differential pairs of outputs for the memory controller.

Chapter 2
Memory Map

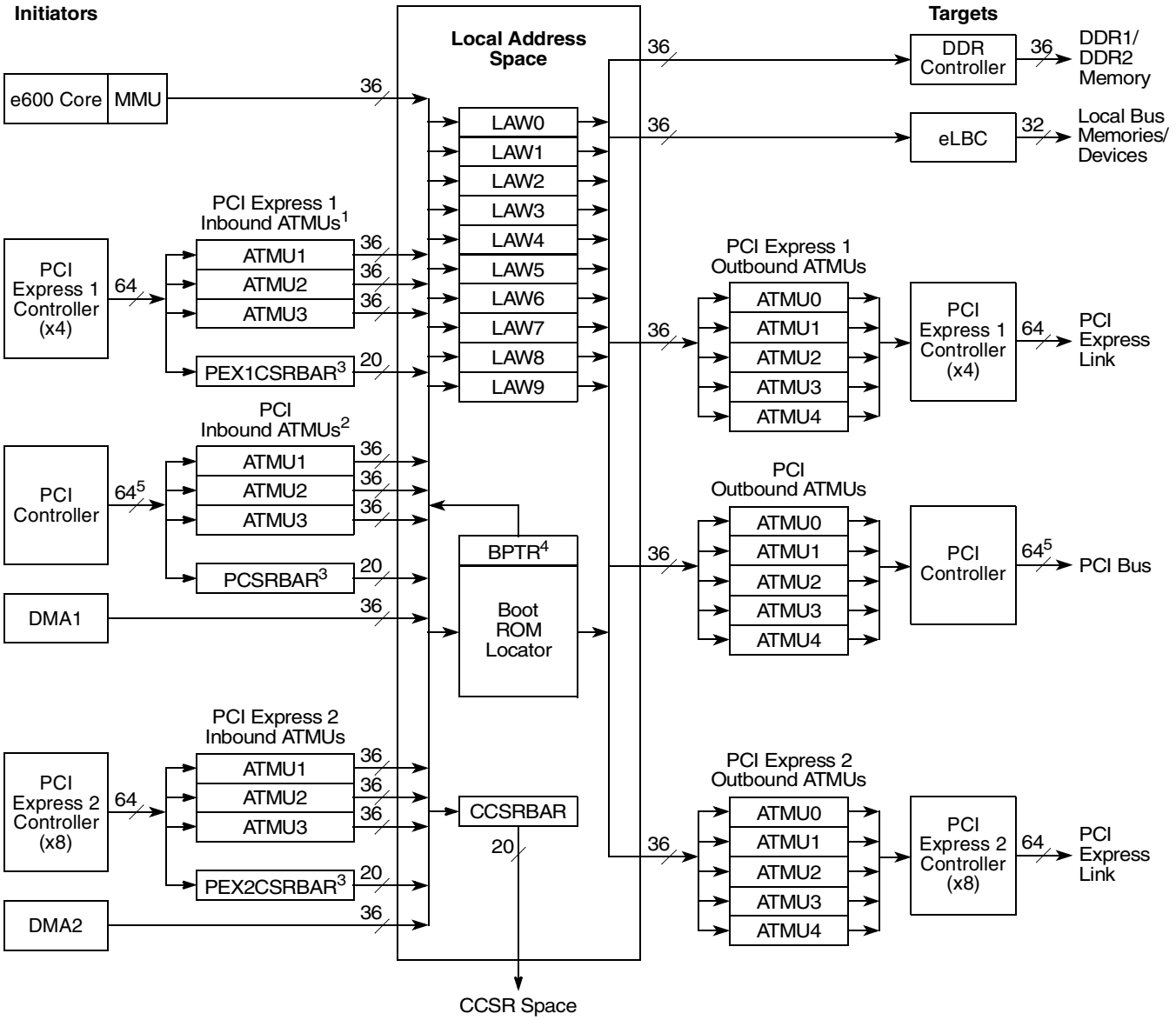
This chapter describes the mechanisms that define the device memory map—the local access windows (LAWs), the address translation and mapping units (ATMUs), and the configuration, control, and status registers (CCSRs).

2.1 Overview

There are several address domains within the MPC8610, including:

- the logical, virtual, and physical (real) address spaces within the e600 core
- the internal local address space
- the internal configuration, control, and status register (CCSR) address space
- the external memory, I/O, and configuration address spaces of the PCI bus
- the external memory, I/O, and configuration address spaces of PCI Express link 1
- the external memory, I/O, and configuration address spaces of PCI Express link 2

Figure 2-1 shows a generalized view of the address domains with potential transaction initiators shown on the left and potential target interfaces shown on the right.



1 The PCI Express 1 inbound ATMU may target the PCI interface across OCN1 without passing through the local address space. However, the ATMU translation must be compatible with any LAW mappings to the PCI controller.

2 The PCI inbound ATMU may target the PCI Express 1 interface across OCN1 without passing through the local address space. However, the ATMU translation must be compatible with any LAW mappings to the PCI Express 1 controller.

3 PEX1CSRBAR, PCSRBAR, and PEX2CSRBAR provide a fixed inbound translation to the 1-Mbyte CCSR space.

4 The boot page translation register (BPTR) is an optional translation from the default boot page to the local address space. BPTR is programmed by the boot sequencer or an external host which must also set up a local access window to route the translated boot address to the proper target.

5 The PCI bus controller is limited to a 32-bit multiplexed address/data bus. However, PCI dual address cycles permit 64-bit addressing, so the address bus is shown here as 64-bits wide.

Figure 2-1. MPC8610 Address Domains

The MMU in the e600 core handles translation of logical (effective) addresses, into virtual addresses, and ultimately to the physical addresses for the local address space. The MMU is described in Section 5.3.5, "Memory Management."

The local address map refers to the physical 36-bit address space seen by the e600 core as it accesses memory and I/O space. The DMA engines also see this same local address map. All memory controlled by the DDR and enhanced local bus controllers exists in this address map, as do all memory-mapped configuration, control, and status registers (CCSRs). The local address map is defined by a set of ten local access windows (LAWs). Each of these windows maps a region of the local address space to a specified target interface, such as the DDR controller, enhanced local bus controller, PCI Express controllers, or other targets. The internal configuration, control, and status registers (CCSRs) for all the functional blocks are located in the local memory space at a specific CCSR window.

If the target mapping performed by the local access windows directs the transaction to one of the external interfaces (as an outbound read or write), the transaction is then mapped into that interface's external address space by address translation and mapping unit (ATMU) windows associated with the external interface. Outbound ATMUs perform the mapping from the local 36-bit address space to the address space of external interface; inbound ATMU windows perform the address translation from the external address space to the local address space.

2.2 Local Access Windows

The local address map is defined by a set of ten local access windows (LAWs). Each of these windows maps a programmable 4-Kbyte to 32-Gbyte region of the local 36-bit address space to a specified target interface, such as the DDR controller, enhanced local bus controller, PCI Express controllers, or other targets. This allows the internal interconnections of the device to route a transaction from its source to the proper target.

Each LAW is defined by a base address register which specifies the starting address for the window, and an attribute register which specifies whether the mapping is enabled, the size of the window, and the target interface for that window. Note that the LAWs do not perform any address translation. and therefore there are no corresponding translation address registers. The local access window registers exist as part of the local access block in the general utilities registers in CCSR space.

With the exception of configuration space (mapped by CCSRBAR) and the default boot ROM space, all addresses used by the system must be mapped by an LAW. This includes addresses that are mapped by inbound ATMU windows. Thus, target mappings of the local access windows and the inbound ATMU windows must be consistent.

2.2.1 Local Access Window Registers

The local access window registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR), plus the block base address, plus the offset of the specific register to be accessed. For the LAWs, the block base address is 0x0_0000.

Note that all LAW registers should only be accessed a word (4-bytes) at a time. Table 2-1 shows the memory map for the LAW registers.

Table 2-1. Local Access Window Memory Map

Offset (Hex)	Register	Access	Reset	Section/Page
Local Access Window Registers—Block Base Address 0x0_0000				
0xC08	LAWBAR0—Local access window 0 base address register	R/W	All zeros	2.2.1.1/2-5
0xC10	LAWAR0—Local access window 0 attribute register	R/W	All zeros	2.2.1.2/2-5
0xC28	LAWBAR1—Local access window 1 base address register	R/W	All zeros	2.2.1.1/2-5
0xC30	LAWAR1—Local access window 1 attribute register	R/W	All zeros	2.2.1.2/2-5
0xC48	LAWBAR2—Local access window 2 base address register	R/W	All zeros	2.2.1.1/2-5
0xC50	LAWAR2—Local access window 2 attribute register	R/W	All zeros	2.2.1.2/2-5
0xC68	LAWBAR3—Local access window 3 base address register	R/W	All zeros	2.2.1.1/2-5
0xC70	LAWAR3—Local access window 3 attribute register	R/W	All zeros	2.2.1.2/2-5
0xC88	LAWBAR4—Local access window 4 base address register	R/W	All zeros	2.2.1.1/2-5
0xC90	LAWAR4—Local access window 4 attribute register	R/W	All zeros	2.2.1.2/2-5
0xCA8	LAWBAR5—Local access window 5 base address register	R/W	All zeros	2.2.1.1/2-5
0xCB0	LAWAR5—Local access window 5 attribute register	R/W	All zeros	2.2.1.2/2-5
0xCC8	LAWBAR6—Local access window 6 base address register	R/W	All zeros	2.2.1.1/2-5
0xCD0	LAWAR6—Local access window 6 attribute register	R/W	All zeros	2.2.1.2/2-5
0xCE8	LAWBAR7—Local access window 7 base address register	R/W	All zeros	2.2.1.1/2-5
0xCF0	LAWAR7—Local access window 7 attribute register	R/W	All zeros	2.2.1.2/2-5
0xD08	LAWBAR8—Local access window 8 base address register	R/W	All zeros	2.2.1.1/2-5
0xD10	LAWAR8—Local access window 8 attribute register	R/W	All zeros	2.2.1.2/2-5
0xD28	LAWBAR9—Local access window 9 base address register	R/W	All zeros	2.2.1.1/2-5
0xD30	LAWAR9—Local access window 9 attribute register	R/W	All zeros	2.2.1.2/2-5

2.2.1.1 Local Access Window *n* Base Address Registers (LAWBAR0–LAWBAR9)

The LAWBARn registers define the 24 high-order address bits that fix the location of each window in the local address space. Note that the minimum size of any LAW is 4 Kbytes, so the 12 lowest-order bits of the base address cannot be specified. Figure 2-2 shows the bit fields of the LAWBAR*n* registers.

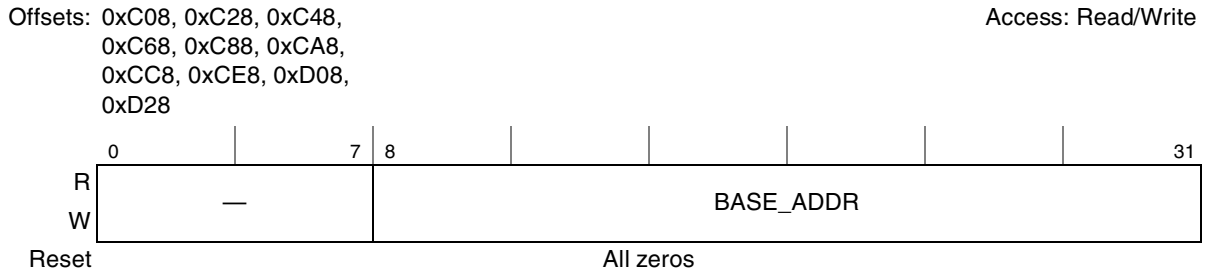


Figure 2-2. Local Access Window Base Address Registers (LAWBAR0 to LAWBAR9)

Table 2-3 describes the LAWBAR*n* fields.

Table 2-2. LAWBAR*n* Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–31	BASE_ADDR	Identifies the 24 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 LAWAR*n*[SIZE].

2.2.1.2 Local Access Window *n* Attributes Registers (LAWAR0–LAWAR9)

The LAWAR*n* registers are used to enable specific local access windows, define their size and specify the target interface. Figure 2-3 shows the bit fields of the LAWAR*n* registers.

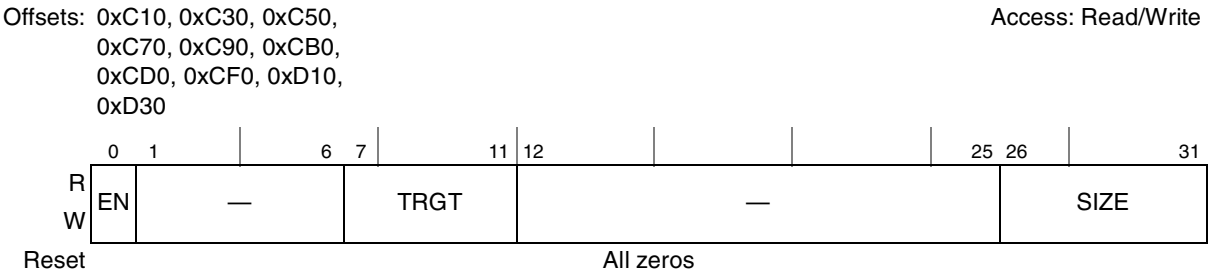


Figure 2-3. Local Access Window Attributes Registers (LAWAR0 to LAWAR9)

Table 2-3 describes the LAWAR*n* fields.

Table 2-3. LAWAR*n* Field Descriptions

Bits	Name	Description
0	EN	0 The local access window *n* (and all other LAWAR*n* and LAWBAR*n* fields) are disabled. 1 The local access window *n* is enabled and other LAWAR*n* and LAWBAR*n* fields combine to identify an address range for this window.
1–6	—	Reserved
7–11	TRGT	Identifies the target interface when a transaction hits in the address range defined by this window. The encodings for TRGT are defined in Table 2-4.
12–25	—	Reserved
26–31	SIZE	Identifies the size of the window from the starting address. Window size is $2^{(SIZE+1)}$ bytes. 000000–001010 Reserved 001011 4 Kbytes 001100 8 Kbytes 001101 16 Kbytes . . . $2^{(SIZE+1)}$ bytes 100010 32 Gbytes 100011–111111 Reserved

The target interface for each LAW is specified using the encodings shown in Table 2-4. Note that configuration registers are mapped by the window defined by CCSRBAR. The CCSR mapping supersedes local access window mappings, so configuration registers do not appear as a target for local access windows.

Table 2-4. Target Interface Encodings

TRGT	Target Interface	TRGT	Target Interface
00000	PCI	10000	Reserved
00001	PCI Express 2 (x8)	10001	Reserved
00010	PCI Express 1 (x4)	10010	Reserved
00011	Reserved	10011	Reserved
00100	Local Bus (eLBC)	10100	Reserved
00101	Reserved	10101	Reserved
00110	Reserved	10110	Reserved
00111	Reserved	10111	Reserved
01000	Reserved	11000	Reserved
01001	Reserved	11001	Reserved
01010	Reserved	11010	Reserved
01011	Reserved	11011	Reserved
01100	Reserved	11100	Reserved
01101	Reserved	11101	Reserved

Table 2-4. Target Interface Encodings (continued)

TRGT	Target Interface	TRGT	Target Interface
01110	Reserved	11110	Reserved
01111	DDR memory controller	11111	Reserved

2.2.2 Precedence of Local Access Windows

If two or more LAWs overlap, the lower-numbered window takes precedence. For instance, consider two LAWs, set up as shown in Table 2-5.

Table 2-5. Overlapping Local Access Windows

LAW	Base Address	Size	Target Interface
1	0x0_7FF0_0000	1 Mbyte	0b00100 (enhanced local bus controller—eLBC)
2	0x0_0000_0000	2 Gbytes	0b01111 (DDR controller)

In this case, LAW 1 governs the mapping of the 1-Mbyte region from 0x0_7FF0_0000 to 0x0_7FFF_FFF, even though the window described in LAW 2 also encompasses that memory region.

NOTE

The CCSR mapping, defined by CCSRBAR, supersedes all local access window mappings.

2.2.3 Configuring Local Access Windows

Once a local access window is enabled, it should not be modified while any device in the system may be using the window. Neither should a new window 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 LAW configuration register before enabling any other devices to use the window. For example, if LAWs 0–3 are being configured in order during the initialization process, the last write (to LAWAR3) should be followed by a read of LAWAR3 before any devices try to use any of these windows. If the configuration is being performed by the e600 core, the read of LAWAR3 should be followed by an **isync** instruction.

2.2.4 Distinguishing Local Access Windows from Other Mapping Functions

It is important to distinguish between the mapping function performed by the LAWs and the additional mapping functions that occur at the target interfaces. The LAWs define how a transaction is routed through the device's internal interconnects from the transaction's source to its target. After the transaction has arrived at its target interface, that interface controller may perform additional mapping. For instance, the DDR controller has chip select registers that map a memory request to a particular external device. Similarly, the enhanced local bus controller has base registers that perform a similar function. The PCI and PCI Express interfaces have outbound address translation and mapping units (ATMUs) that map the local address into an external address space.

These other mapping functions are configured by programming the CCSRs of the individual interfaces. Note that there is no need to have a one-to-one correspondence between LAWs and chip select regions or outbound ATMU windows. A single LAW can be further decoded to any number of chip selects or to any number of outbound ATMU windows at the target interface.

2.2.5 Illegal Interaction Between Local Access Windows and DDR Chip Selects

If a local access window maps an address to an interface other than the DDR controllers, then there should not be a valid chip select configured for the same address in the DDR controller. Because DDR chip select boundaries are defined by a beginning and ending address, it is easy to define them so that they do not overlap with LAWs that map to other interfaces.

2.2.6 Local Address Map Example

Figure 2-4 shows what a typical local address map might look like.

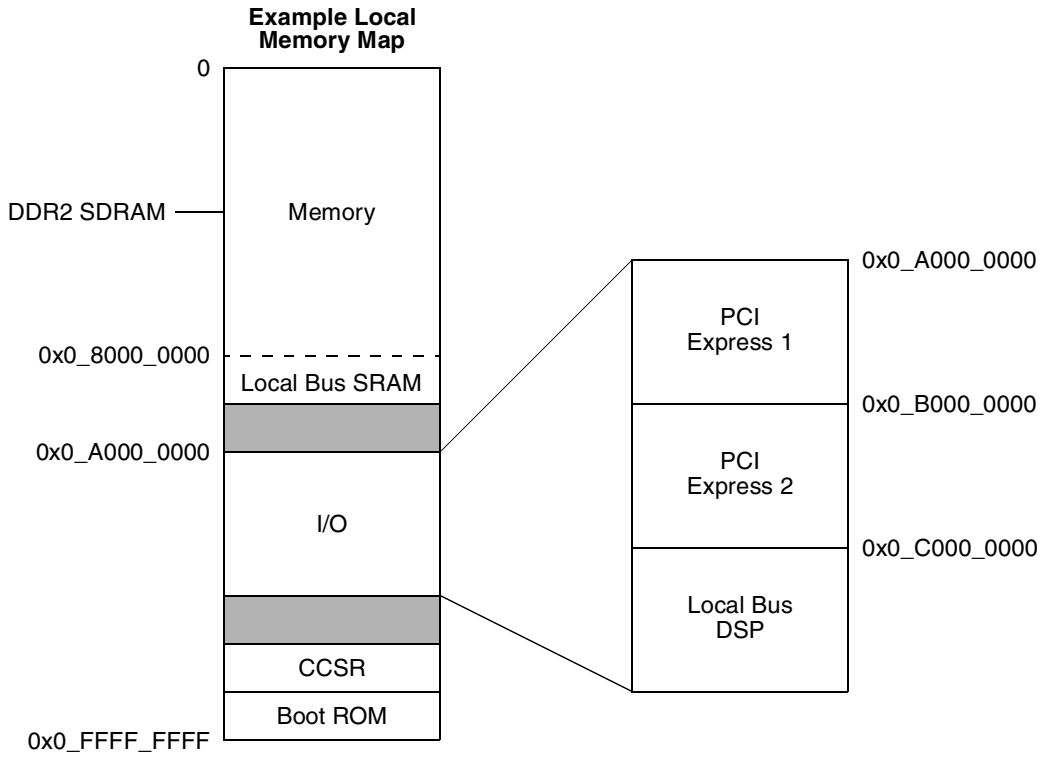


Figure 2-4. Local Address Map Example

Table 2-6 shows the corresponding set of LAW settings for the example shown in Figure 2-4.

Table 2-6. Local Access Window Settings Example

Window	Base Address	SIZE	Target Interface	TRGT
0	0x0_0000_0000	2 Gbytes	DDR	0b01111
1	0x0_8000_0000	1 Mbyte	Local bus controller (eLBC)—SRAM	0b00100
2	0x0_A000_0000	256 Mbytes	PCI Express 1	0b00001
3	0x0_B000_0000	256 Mbytes	PCI Express 2	0b00010
4	0x0_C000_0000	256 Mbytes	Local bus controller (eLBC)—DSP	0b00100
5–9	Unused			

In this example, it is not necessary to use a LAW to specify the location of the boot ROM because it is in the default location at the 8 Mbytes of memory from 0x0_FF80_0000 to 0x0_FFFF_FFFF (see Section 4.4.3.8, “Boot ROM Location”); the e600 core fetches its reset vector by performing a burst read from effective address 0x0_FFF0_0100 and begins executing with the instruction at effective address 0x0_FFF0_0100. Neither is it required to define a LAW to describe the range of memory used for memory-mapped configuration, control, and status registers because these occupy a fixed 1-Mbyte space pointed to by CCSRBAR. See Section 4.3.1.1.3, “Configuration, Control, and Status Base Address Register (CCSRBAR).”

2.3 Address Translation and Mapping Units

To facilitate flexibility in defining the address maps for the external interfaces (PCI, PCI Express 1, and PCI Express 2), the device provides address translation and mapping units (ATMUs).

The following types of translation and mapping operations are performed by the ATMUs:

- Translating the local 36-bit address to an external address space
- Translating external addresses to the local 36-bit address space
- Assigning attributes to transactions
- Mapping a local address to a target interface

Outbound address translation and mapping refers to the translation of addresses from the local 36-bit address space to the external address space and attributes of a particular I/O interface.

Inbound address translation and mapping refers to the translation of an address from the external address space of an I/O interface to the local address space understood by the internal interfaces of this device. It also refers to the mapping of transactions to a particular target interface and the assignment of transaction attributes. Note that in mapping the transaction to the target interface, an inbound ATMU window performs a function similar to that of the local access windows. The target mappings created by an inbound ATMU must be consistent with those of the LAWs. That is, if an inbound ATMU maps a transaction to a given local address and a given target, a LAW must also map that same local address to the same target.

2.3.1 Address Translation

All of the configuration registers that define an ATMU window's translation and mapping functions follow the same general register format, summarized in Table 2-7.

Table 2-7. Format of ATMU Window Definitions

Register	Function
Translation address (TAR)	High-order address bits defining location of the window in the target address space
Base address (BAR)	High-order address bits defining location of the window in the initiator address space
Window attributes (WAR)	Window enable, window size, target interface, and transaction attributes

The size of the windows must be a power-of-two. To perform a translation or 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 a window, if address translation is being performed, the new translated address is created by concatenating the window offset to the translation address. Again, the window's size attribute dictates how many bits are translated.

2.3.2 Outbound ATMUs

If the target mapping performed by the local access windows directs the transaction to one of the external interfaces (as an outbound read or write), the transaction is then mapped into that interface's external address space by outbound ATMUs associated with the external interface. The outbound ATMUs perform the mapping from the local 36-bit address space to the address spaces of the PCI Express and PCI busses. which may be much larger than the local space. The outbound ATMUs also map attributes such as transaction type and priority level.

The PCI controller and PCI Express controllers each have four outbound ATMU windows plus a default window. If a transaction's address does not hit any of the four outbound ATMU windows, the translation attributes defined by the default window are used. The default window is always enabled. The outbound ATMU registers include extended translation address registers so that up to 64 bits of external address space can be supported.

2.3.3 Inbound ATMUs

The inbound ATMUs perform the address translation from the external address spaces to the local address space, attach attributes and transaction types to the transaction, and also map the transaction to its target interface.

The PCI and PCI Express controllers each have three general inbound ATMU windows plus a fixed translation window to the CCSR space.

2.3.3.1 Illegal Interaction Between Inbound ATMUs and LAWs

Since both local access windows and inbound ATMUs map transactions to a target interface, it is essential that they not contradict one another. For example, it is considered a programming error to have an inbound ATMU map a transaction to the DDR memory controller (target interface 0b0_1111) if the resulting

translated local address is mapped to the PCI interface (target interface 0b0_0000) by a local access window. Such programming errors may result in unpredictable system deadlocks.

2.4 Configuration, Control, and Status Registers

All of the memory-mapped configuration, control, and status registers (CCSRs) in this device are contained within a 1-Mbyte address region. To allow for flexibility, the CCSR block is relocatable in the local address space. With some exceptions, the CCSRs must be accessed 32-bits at a time. Notable exceptions are the global timer module, the infrared controller, and the watchdog timer module registers, and the PCI and PCI Express configuration data ports (CFG_DATA and PEX*n*_CONFIG_DATA), which support 32-bit, 16-bit, or 8-bit accesses. The DUART and I2C registers must be accessed as single bytes only.

The local address map location of the CCSR block is controlled by the configuration, control, and status base address register (CCSRBAR); see Section 4.3.1.1.3, “Configuration, Control, and Status Base Address Register (CCSRBAR).” The default value for CCSRBAR is 0x0_FF70_0000 (or 4 Gbytes-9 Mbytes). No address translation is performed for CCSR space, so there is no associated translation address register. The CCSR window is always enabled with a fixed size of 1 Mbyte; no other attributes are attached, so there is no associated window attribute register.

NOTE

Note that the CCSR window always takes precedence over all local access windows. However, the CCSR window must not overlap an LAW that maps to the DDR controller. Otherwise, undefined behavior occurs.

An example of a top-level memory map with the default location of the configuration, control, and status registers is shown in Figure 2-5.

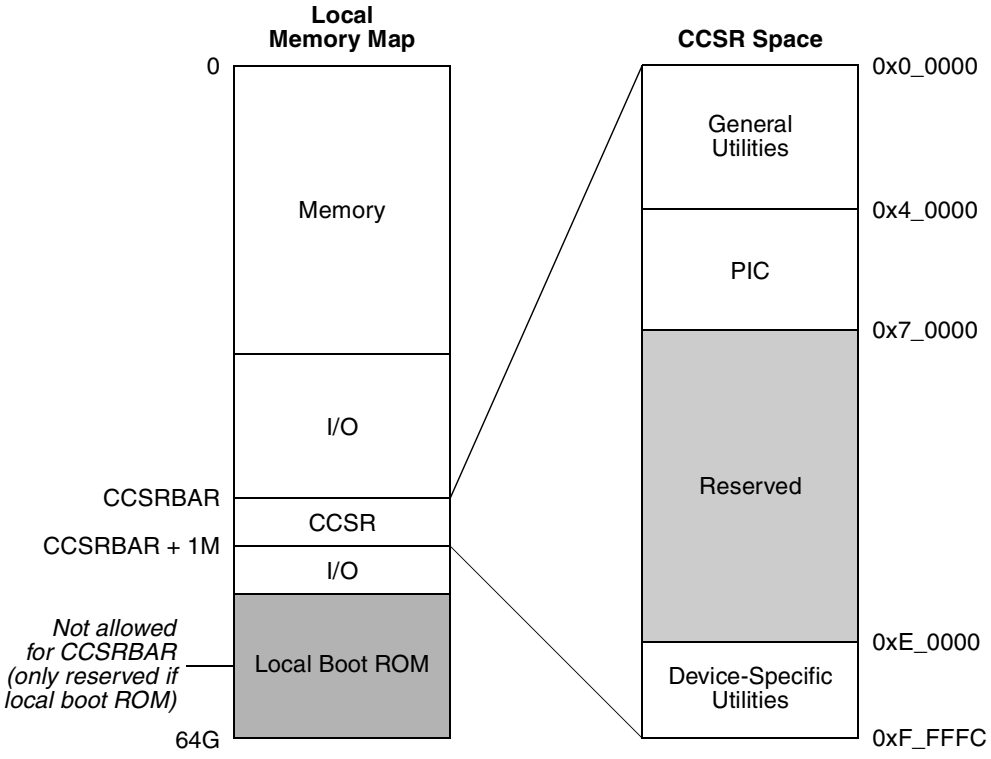


Figure 2-5. CCSR Space

2.4.1 Accessing CCSR Memory from the Local Processor

When the local e600 core is used to configure CCSR space, the CCSR memory space should be marked as cache-inhibited and guarded.

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

To guarantee that the results of any sequence of writes to configuration registers are in effect, the final configuration register write should be chased 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.

2.4.2 Accessing CCSR Memory from External Masters

In addition to being accessible by the e600 processor, the CCSRs are accessible from the external interfaces—PCI, PCI Express 1, and PCI Express 2. This allows external masters on the I/O ports to configure the device.

External masters do not need to know the location of the CCSR memory in the local address map. Rather, they access the CCSR 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 CCSR memory is selectable through the PCI configuration and status register base address register (PCSRBAR), at offset 0x10. An external PCI master sets this register by performing a PCI configuration cycle to this device. Subsequent memory accesses by a PCI master to the PCI address range indicated by PCSRBAR are translated to the local CCSR address indicated by the current setting of CCSRBAR.

Similar to PCI, each PCI Express controller has a base address for accessing the local CCSR block—the PCI Express configuration and status register base address register (PEXCSRBAR), at offset 0x10 in PCI Express configuration space. An external PCI Express master sets this register by performing a PCI Express configuration cycle to this device. Subsequent memory accesses by a PCI Express master to the PCI Express address range indicated by PEXCSRBAR are translated to the local CCSR address indicated by the current setting of CCSRBAR.

2.4.3 Organization of CCSR Space

As shown in Figure 2-5, the CCSR space is divided into three groups—general utilities, programmable interrupt controller (PIC), and device-specific utilities registers.

2.4.3.1 General Utilities Registers

The general utilities registers are the functional block-specific registers that occupy the first 256 Kbytes of CCSR space. Each functional block is allocated a 4-Kbyte address range for its registers within the general utilities space. Figure 2-6 shows the layout of the general utilities registers.

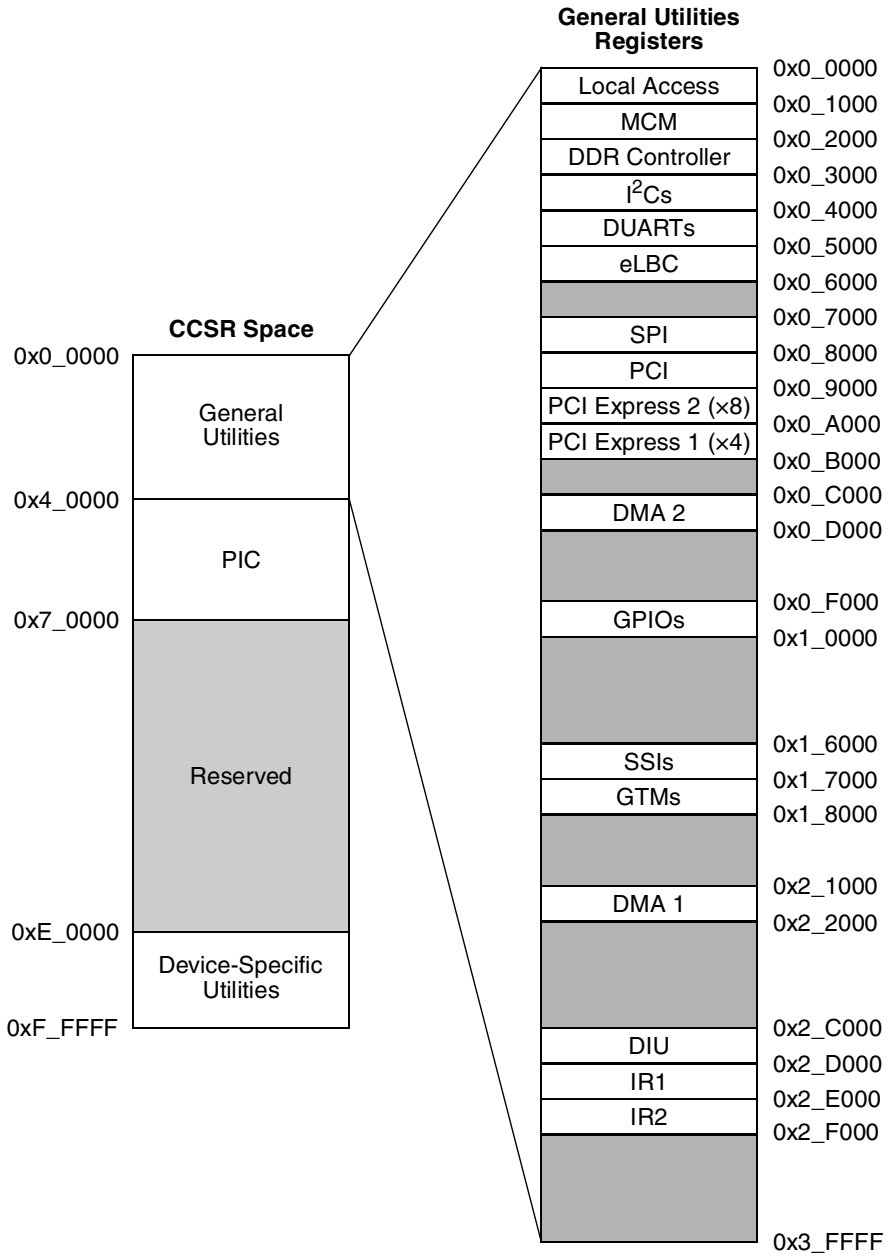


Figure 2-6. General Utilities Register Map within CCSR Space

2.4.3.1.1 General Utilities Register Organization

Figure 2-7 shows the typical organization of registers inside the 4-Kbyte register space allocated to an individual functional block. Starting at the block base address, the first 3 Kbytes are available for general registers. If the functional block has associated ATMUs, the next 512 bytes are dedicated to address translation and mapping registers. If a functional block has error management registers, they are typically placed starting at offset 0xE00 from the block base address, and any debug registers are typically placed in the final 256 bytes of the block's register space starting at offset 0xF00.

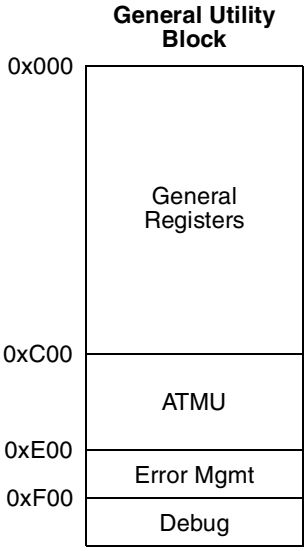


Figure 2-7. General Utility Register Block

NOTE

Refer to detailed register descriptions for each functional block for exact locations, sizes, and access requirements.

2.4.3.2 Programmable Interrupt Controller Registers

The programmable interrupt controller (PIC) follows the OpenPIC programming model which requires a larger register address space than the 4 Kbytes allocated to other blocks within the general utilities space. For this reason, the PIC is allocated the second 256 Kbytes of CCSR space, beginning at offset 0x4_0000 from CCSRBAR. The layout of the PIC register space is shown in Figure 2-8. Note that the PIC registers should only be accessed with 32-bit accesses.

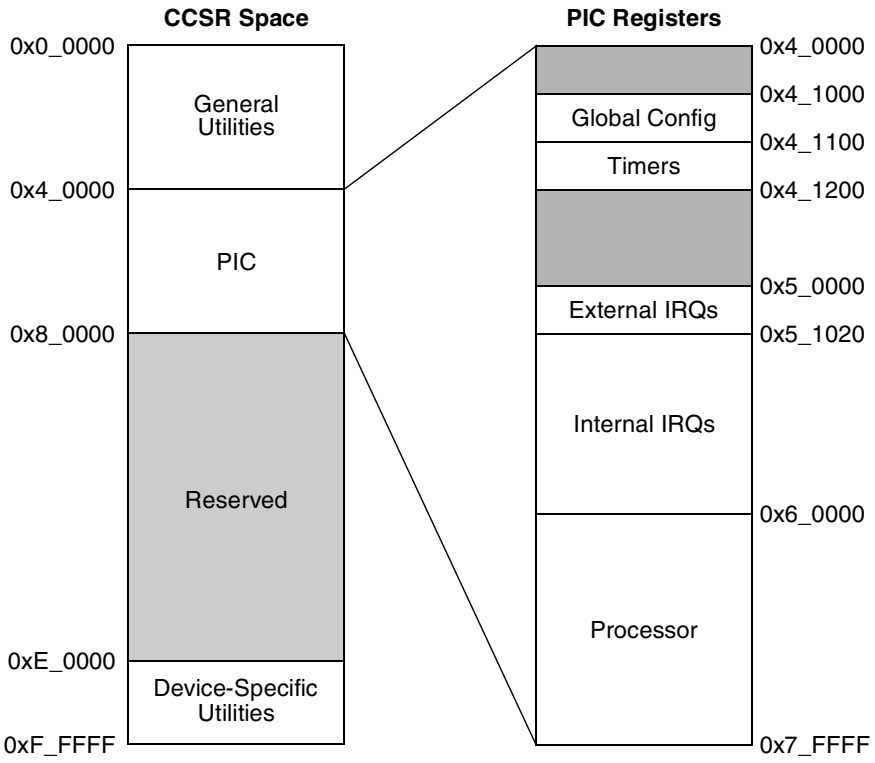


Figure 2-8. PIC Register Map within CCSR Space

2.4.3.3 Device-Specific Utilities Registers

The device-specific utilities registers control functions that are not particular to a functional unit but to the device as a whole; they occupy the highest 256 Kbytes of CCSR space. The device-specific utilities registers consist of power management, performance monitors, and device-wide debug utilities. Figure 2-9 shows the layout of the device-specific utilities registers. Note that the device-specific registers are accessible with 32-bit accesses only. Transactions of a size other than 32-bits are considered programming errors and the operation is undefined.

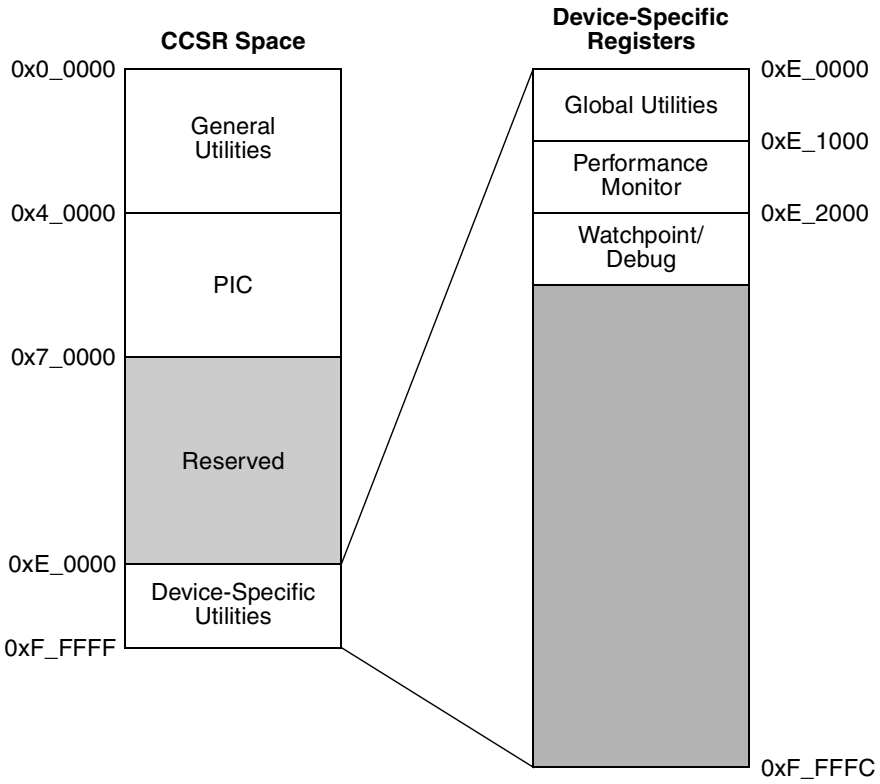


Figure 2-9. Device-Specific Register Map within CCSR Space

2.4.4 CCSR Address Map

The full register address of any CCSR is comprised of the CCSR window base address, specified in CCSRBAR (default address 0x0_FF70_0000), plus the functional block base address, plus the specific register's offset within that block.

Table 2-8 shows the location of the functional block base addresses for the entire CCSR space. Cross-references are provided to the CCSR maps for each individual block. A complete listing of all CCSRs is provided in Appendix A, "Complete List of Configuration, Control, and Status Registers."

Table 2-8. CCSR Block Base Address Map

Block Base Address (Hex)	Block	Section/Page	Comments
General Utilities (0x0_0000–0x3_FFFF)			
0x0_0000	Local configuration control	4.3.1/4-3	—
	Local access	2.2.1/2-3	—
0x0_1000	MPX coherency module (MCM)	7.4/7-4	—
0x0_2000	DDR memory controller	8.4/8-9	—
0x0_3000	I^2C controllers (2)	12.4/12-4	I^2C controller 1: 0x0_3000 I^2C controller 2: 0x0_3100

Table 2-8. CCSR Block Base Address Map (continued)

Block Base Address (Hex)	Block	Section/Page	Comments
0x0_4000	DUARTs (2)	13.3/13-5	UART0: 0x0_4500 (DUART1) UART1: 0x0_4600 (DUART1) UART2: 0x0_4700 (DUART2) UART3: 0x0_4800 (DUART2)
0x0_5000	Enhanced local bus controller (eLBC)	9.3/9-8	—
0x0_6000– 0x0_6FFF	Reserved	—	—
0x0_7000	Serial peripheral interface (SPI)	15.4/15-8	—
0x0_8000	PCI controller	20.3/20-11	—
0x0_9000	PCI Express controller 2 (x8)	21.3/21-5	—
0x0_A000	PCI Express controller 1 (x4)	21.3/21-5	—
0x0_B000– 0x0_BFFF	Reserved	—	—
0x0_C000	DMA controller 2	19.3/19-6	DMA controller 1 is located at block base 0x2_1000
0x0_D000– 0x0_EFFF	Reserved	—	—
0x0_F000	GPIO controllers (2)	22.5/22-3	GPIO1: 0x0_F000 GPIO2: 0x0_F100
0x1_0000– 0x1_5FFF	Reserved	—	—
0x1_6000	Synchronous serial interface (SSI) controllers (2)	16.3/16-23	SSI1: 0x1_6000 SSI2: 0x1_6100
0x1_7000	Global timer modules (2)	17.3/17-5	GTM1: 0x1_7000 GTM2: 0x1_7100
0x1_8000– 0x2_0FFF	Reserved	—	—
0x2_1000	DMA controller 1	19.3/19-6	DMA controller 2 is located at block base 0x0_C000
0x2_2000– 0x2_BFFF	Reserved	—	—
0x2_C000	Display interface unit (DIU)	10.3/10-3	—
0x2_D000	Infrared interface controller 1	14.4/14-4	SIRI1: 0x2_D000 FIRI1: 0x2_D100
0x2_E000	Infrared interface controller 2	14.4/14-4	SIRI2: 0x2_E000 FIRI2: 0x2_E100
0x2_F000– 0x3_FFFF	Reserved	—	—

Table 2-8. CCSR Block Base Address Map (continued)

Block Base Address (Hex)	Block	Section/Page	Comments
Programmable Interrupt Controller (PIC) (0x4_000–0x6_FFFF)			
0x4_0000	PIC—Global registers	11.3/11-9	Gbl config: 0x4_1000 Gbl timers: 0x4_1100
0x5_0000	PIC—Interrupt source registers	11.3/11-9	External IRQs: 0x5_0000 Internal IRQs: 0x5_1200
0x6_0000	PIC—Processor (core) registers	11.3/11-9	—
Reserved (0x7_0000–0xD_FFFF)			
Device-Specific Utilities (0xE_0000–0xF_FFFF)			
0xE_0000	Global utilities	23.4/23-3	—
0xE_1000	Performance monitor	24.3/24-3	—
0xE_2000	Watchpoint monitor and trace buffer	25.3/25-7	—
0xE_3000– 0xE_3FFF	Reserved	—	—
0xE_4000	Watchdog timer	18.2/18-2	—
0xE_5000– 0xF_FFFF	Reserved	—	—

Chapter 3
Signal Descriptions

This chapter describes the MPC8610 external signals. It is organized into the following sections:

- Overview of signals and cross-references for signals serving multiple functions, including two lists:
  - by functional block
  - 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.

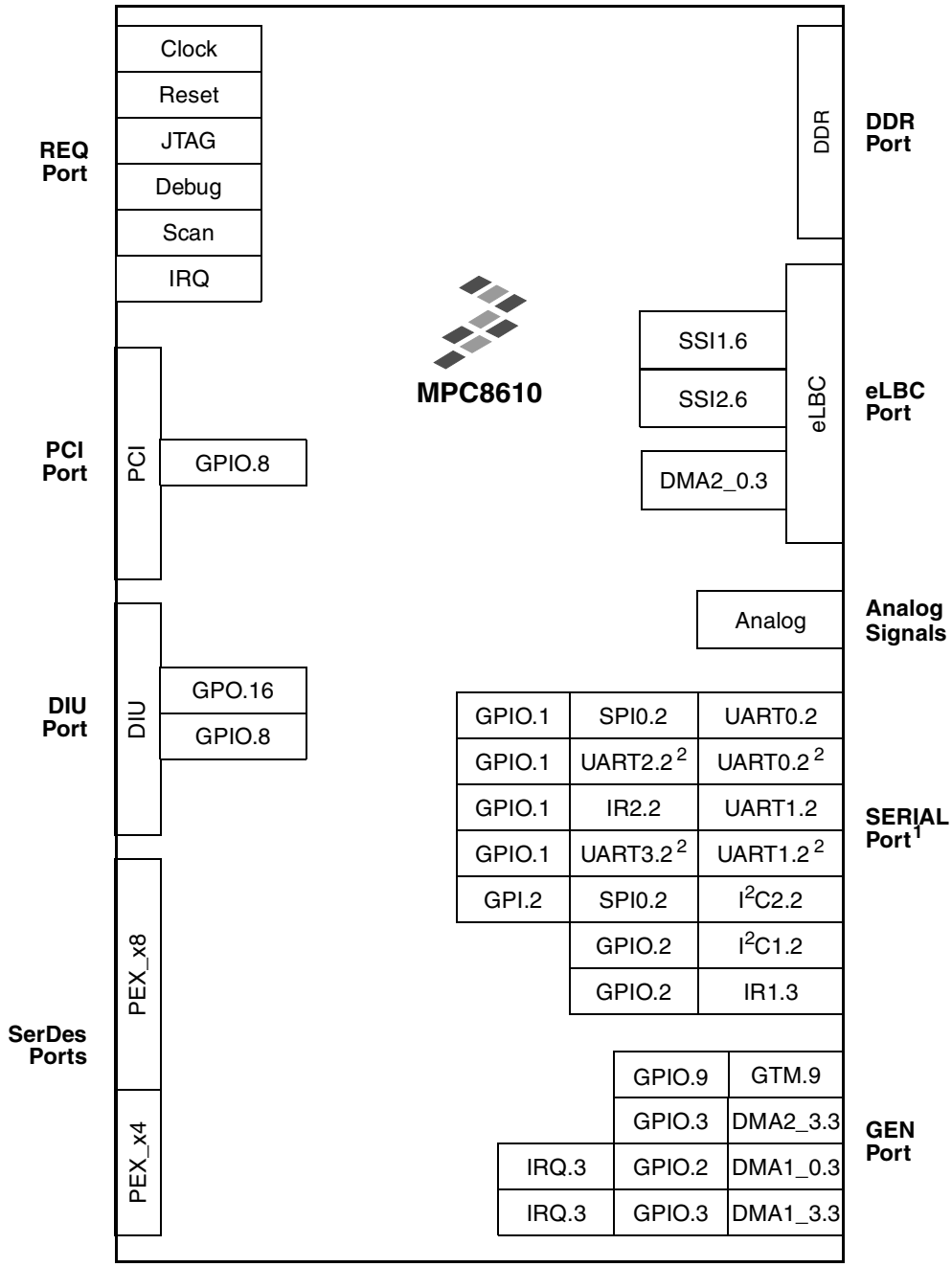
3.1 Signals Overview

The MPC8610 signals are grouped as follows:

- Clock signals
- DDR memory interface signals
- Enhanced local bus interface signals
- LCD interface signals
- PIC interface signals
- Dual I^2C interface signals
- Dual DUART interface signals
- IrDA interface signals
- SPI interface signals
- SSI interface signals
- Synchronous serial interface signals
- DMA interface signals

- General-purpose timer signals
- PCI interface signals
- PCI Express interface signals
- General purpose I/O signals
- System control, power management, and debug signals
- Test, JTAG, and configuration signals

Figure 3-1 illustrates the external signals of the MPC8610, showing how the signals are grouped; individual signals are listed in Table 3-1 and Table 3-2. Refer to the *MPC8610 Integrated Processor Hardware Specifications* for a pinout diagram showing pin numbers and a listing of all the electrical and mechanical specifications.



1 UART0 and UART1 are in DUART1; UART2 and UART3 are in DUART2. When both DUARTs are used, the $\overline{\text{RTS}}$ and $\overline{\text{CTS}}$ signals of DUART1 are not available.

2 These pins carry either the $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ signals of DUART1 (UART0/UART1) or the SIN/SOUT signals of DUART2 (UART2/UART3). Other signals may be shared as well; see Table 3-2 for details.

Figure 3-1. MPC8610 Signal Groupings

The following tables provide summaries of signal functionality on the device. Table 3-1 provides a summary of the signals grouped by function, and Table 3-2 provides the summary list 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.

Note that the direction of the multiplexed signals applies for the primary signal function listed in the left-most column of the table for that row (and does not apply for the state of the reset configuration signals). Finally, the table provides a pointer to the table where the signal function is described.

Table 3-1. MPC8610 Signal Reference by Functional Block

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
System Clocking Signals						
SYSCLK	System clock/PCI clock	Clock	—	1	I	4-3/4-3
RTC	Real time clock	Clock	—	1	I	4-3/4-3
DDR Memory Interface Signals						
MA[15:0]	Address bus	DDR Memory	—	16	O	8.3/8-3
MBA[2:0]	Logical bank address	DDR Memory	—	3	O	8.3/8-3
$\overline{\text{MCS}}$[0:3]	Chip selects	DDR Memory	—	4	O	8.3/8-3
MDQ[0:63]	Data bus	DDR Memory	—	64	I/O	8.3/8-3
MECC[0:7]	Error checking and correcting	DDR Memory	—	8	I/O	8.3/8-3
MDM[0:8]	Data mask	DDR Memory	—	9	O	8.3/8-3
MDQS[0:8]	Data strobes	DDR Memory	—	9	I/O	8.3/8-3
$\overline{\text{MDQS}}$[0:8]	Complement data strobes	DDR Memory	—	9	I/O	8.3/8-3
$\overline{\text{MCAS}}$	Column address strobe	DDR Memory	—	1	O	8.3/8-3
$\overline{\text{MWE}}$	Write enable	DDR Memory	—	1	O	8.3/8-3
$\overline{\text{MRAS}}$	Row address strobe	DDR Memory	—	1	O	8.3/8-3
MCK[0:5]	DRAM clock outputs	DDR Memory	—	6	O	8.3/8-3
$\overline{\text{MCK}}$[0:5]	DRAM clock outputs (complement)	DDR Memory	—	6	O	8.3/8-3
MCKE[0:3]	DRAM clock enable	DDR Memory	—	4	O	8.3/8-3
MDIC[0:1]	Driver impedance calibration	DDR Memory	—	2	I/O	8.3/8-3
MODT[0:3]	DRAM on-die termination	DDR Memory	—	4	O	8.3/8-3

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
Enhanced Local Bus Signals						
LAD[0:31]	muxed data/address	eLBC	cfg_gpinput[0:31]	32	I/O	9.2/9-4
LDP[0:3]/LA[6:9]	Data parity/non-multiplexed address bus	eLBC	—	4	I/O	9.2/9-4
LA10	Non-multiplexed address bus	eLBC	SSI1_TXD/ cfg_ssi_la_sel	1	O	9.2/9-4
LA11	Non-multiplexed address bus	eLBC	SSI1_TFS	1	O	9.2/9-4
LA12	Non-multiplexed address bus	eLBC	SSI1_TCK	1	O	9.2/9-4
LA13	Non-multiplexed address bus	eLBC	SSI1_RCK	1	O	9.2/9-4
LA14	Non-multiplexed address bus	eLBC	SSI1_RFS	1	O	9.2/9-4
LA15	Non-multiplexed address bus	eLBC	SSI1_RXD	1	O	9.2/9-4
LA16	Non-multiplexed address bus	eLBC	SSI2_TXD	1	O	9.2/9-4
LA17	Non-multiplexed address bus	eLBC	SSI2_TFS	1	O	9.2/9-4
LA18	Non-multiplexed address bus	eLBC	SSI2_TCK	1	O	9.2/9-4
LA19	Non-multiplexed address bus	eLBC	SSI2_RCK	1	O	9.2/9-4
LA20	Non-multiplexed address bus	eLBC	SSI2_RFS	1	O	9.2/9-4
LA21	Non-multiplexed address bus	eLBC	SSI2_RXD	1	O	9.2/9-4
LA[22:24]	Non-multiplexed address bus	eLBC	—	3	O	9.2/9-4
LA25	Non-multiplexed address bus	eLBC	cfg_elbc_clkdiv0	1	O	9.2/9-4
LA26	Non-multiplexed address bus	eLBC	cfg_elbc_clkdiv1	1	O	9.2/9-4
LA27	Non-multiplexed address bus	eLBC	cfg_cpu_boot	1	O	9.2/9-4
LA[28:31]	Non-multiplexed address bus	eLBC	cfg_sys_pll[1:4]	4	O	9.2/9-4
$\overline{\text{LCS}}$[0:4]	Chip selects	eLBC	—	5	O	9.2/9-4
$\overline{\text{LCS5}}$	Chip Selects / DMA Triggers	eLBC	$\overline{\text{DMA2\_DREQ0}}$	1	O	9.2/9-4
$\overline{\text{LCS6}}$	Chip Selects / DMA Triggers	eLBC	$\overline{\text{DMA2\_DACK0}}$	1	O	9.2/9-4
$\overline{\text{LCS7}}$	Chip Selects / DMA Triggers	eLBC	$\overline{\text{DMA2\_DDONE0}}$	1	O	9.2/9-4
$\overline{\text{LWE0}}$/$\overline{\text{LFWE}}$/ $\overline{\text{LBS0}}$	Write enable 0/ LFWE/byte select 0	eLBC	cfg_pci_speed	1	O	9.2/9-4
$\overline{\text{LWE1}}$/$\overline{\text{LBS1}}$	Write enable 1/byte select 1	eLBC	cfg_host_agt0	1	O	9.2/9-4
$\overline{\text{LWE2}}$/$\overline{\text{LBS2}}$	Write enable 2/byte select 2	eLBC	cfg_host_agt1	1	O	9.2/9-4
$\overline{\text{LWE3}}$/$\overline{\text{LBS3}}$	Write enable 3/byte select 3	eLBC	cfg_host_agt2	1	O	9.2/9-4
LBCTL	Buffer control	eLBC	cfg_core_pll0	1	O	9.2/9-4
LALE	Address latch enable	eLBC	cfg_core_pll1	1	O	9.2/9-4

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
LGPL0/ LFCLE	UPM general-purpose line 0/ FCM flash command latch enable	eLBC	cfg_net2_div	1	O	9.2/9-4
LGPL1/ LFALE	UPM general-purpose line 1/ FCM flash address latch enable	eLBC	cfg_pci_clk	1	O	9.2/9-4
LGPL2/ $\overline{\text{LOE}}$/ $\overline{\text{LFRE}}$	UPM general-purpose line 2 / GPCM output enable/FCM flash read enable	eLBC	cfg_core_pll2	1	O	9.2/9-4
LGPL3/ $\overline{\text{LFWP}}$	UPM general-purpose line 3 / FCM flash write protect	eLBC	cfg_boot_seq0	1	O	9.2/9-4
$\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/ LGPL4/LUPWAIT/ LPBSE	GPCM target ack/ FCM flash ready(busy)/ UPM general-purpose line 4/ UPM wait/parity byte select	eLBC	—	1	I/O	9.2/9-4
LGPL5	UPM general-purpose line 5	eLBC	cfg_boot_seq1	1	O	9.2/9-4
LCLK[0:2]	Local bus clocks	eLBC	—	3	O	9.2/9-4
LCD Signals						
DIU_LD[23:16]	DIU data (Red[7:0])	LCD	GPIO1[15:8]	8	O	10.2/10-2
DIU_LD15	DIU data (Green7)	LCD	GPIO1[31]/ cfg_rom_loc3	1	O	10.2/10-2
DIU_LD14	DIU data (Green6)	LCD	GPIO1[30]/ cfg_rom_loc2	1	O	10.2/10-2
DIU_LD13	DIU data (Green5)	LCD	GPIO1[29]/ cfg_rom_loc1	1	O	10.2/10-2
DIU_LD12	DIU data (Green4)	LCD	GPIO1[28]/ cfg_dram_type1	1	O	10.2/10-2
DIU_LD11	DIU data (Green3)	LCD	GPIO1[27]/ cfg_dram_type0	1	O	10.2/10-2
DIU_LD10	DIU data (Green2)	LCD	GPIO1[26]/ cfg_sys_pll0	1	O	10.2/10-2
DIU_LD9	DIU data (Green1)	LCD	GPIO1[25]/ cfg_io_ports2	1	O	10.2/10-2
DIU_LD8	DIU data (Green0)	LCD	GPIO1[24]/ cfg_io_ports1	1	O	10.2/10-2
DIU_LD7	DIU data (Blue7)	LCD	GPIO1[23]/ cfg_io_ports0	1	O	10.2/10-2
DIU_LD6	DIU data (Blue6)	LCD	GPIO1[22]/ cfg_lx2_lockov	1	O	10.2/10-2
DIU_LD5	DIU data (Blue5)	LCD	GPIO1[21]/ cfg_lx1_lockov	1	O	10.2/10-2

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
DIU_LD4	DIU data (Blue4)	LCD	GPIO1[20]/ cfg_core_pll3	1	O	10.2/10-2
DIU_LD3	DIU data (Blue3)	LCD	GPIO1[19]	1	O	10.2/10-2
DIU_LD2	DIU data (Blue2)	LCD	GPIO1[18]	1	O	10.2/10-2
DIU_LD1	DIU data (Blue1)	LCD	GPIO1[17]	1	O	10.2/10-2
DIU_LD0	DIU data (Blue0)	LCD	GPIO1[16]/ cfg_elbc_ecc	1	O	10.2/10-2
DIU_VSYNC	DIU vertical sync	LCD	cfg_pci_impd	1	O	10.2/10-2
DIU_HSYNC	DIU horizontal sync	LCD	cfg_pci_arb	1	O	10.2/10-2
DIU_DE	DIU data enable	LCD	cfg_rom_loc0	1	O	10.2/10-2
DIU_CLK_OUT	DIU pixel clock	LCD	—	1	O	10.2/10-2
Programmable Interrupt Controller (PIC) Signals						
IRQ[0:5]	External interrupt 0–5	PIC	—	6	I	11.2/11-7
IRQ6	External interrupt 6	PIC	$\overline{\text{DMA1\_DREQ0}}$	1	I	11.2/11-7
IRQ7	External interrupt 7	PIC	$\overline{\text{DMA1\_DACK0}}$	1	I	11.2/11-7
IRQ8	External interrupt 8	PIC	$\overline{\text{DMA1\_DDONE0}}$	1	I	11.2/11-7
IRQ9	External interrupt 9	PIC	$\overline{\text{DMA1\_DREQ3}}$	1	I	11.2/11-7
IRQ10	External interrupt 10	PIC	$\overline{\text{DMA1\_DACK3}}$	1	I	11.2/11-7
IRQ11	External interrupt 11	PIC	$\overline{\text{DMA1\_DDONE3}}$	1	I	11.2/11-7
$\overline{\text{IRQ\_OUT}}$	Interrupt output	PIC	—	1	O	11.2/11-7
$\overline{\text{MCP}}$	Machine check	PIC	—	1	I	
I^2C Signals						
IIC1_SDA	I^2C serial data	I^2C	GPIO2[10]	1	I/O	12.3/12-3
IIC1_SCL	I^2C serial clock	I^2C	GPIO2[9]	1	I/O	12.3/12-3
IIC2_SDA	I^2C serial data	I^2C	GPIO2[12]/$\overline{\text{SPISEL}}$	1	I/O	12.3/12-3
IIC2_SCL	I^2C serial clock	I^2C	GPIO2[11]/SPICLK	1	I/O	12.3/12-3
DUART Signals						
UART_SIN0	DUART serial data in	DUART 1	SPIMOSI/GPIO2[5]	1	I	13.2/13-4
UART_SOUT0	DUART serial data out	DUART 1	SPIMISO	1	O	13.2/13-4
$\overline{\text{UART\_CTS0}}$	DUART clear to send	DUART 1	GPIO2[6]/ UART_SIN2	1	I	13.2/13-4
$\overline{\text{UART\_RTS0}}$	DUART ready to send	DUART 1	cfg_wdt_en/ UART_SOUT2	1	O	13.2/13-4

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
UART_SIN1	DUART serial data in	DUART 1	IR2_RXD/GPIO2[7]	1	I	13.2/13-4
UART_SOUT1	DUART serial data out	DUART 1	IR2_TXD	1	O	13.2/13-4
$\overline{\text{UART\_CTS1}}$	DUART clear to send	DUART 1	GPIO2[8]/ UART_SIN3	1	I	13.2/13-4
$\overline{\text{UART\_RTS1}}$	DUART ready to send	DUART 1	UART_SOUT3	1	O	13.2/13-4
UART_SIN2	DUART serial data in	DUART 2	GPIO2[6]/ $\overline{\text{UART\_CTS0}}$	1	I	13.2/13-4
UART_SOUT2	DUART serial data out	DUART 2	cfg_wdt_en/ $\overline{\text{UART\_RTS0}}$	1	O	13.2/13-4
UART_SIN3	DUART serial data in	DUART 2	GPIO2[8]/ $\overline{\text{UART\_CTS1}}$	1	I	13.2/13-4
UART_SOUT3	DUART serial data out	DUART 2	$\overline{\text{UART\_RTS1}}$	1	O	13.2/13-4
IrDA Signals						
IR1_TXD	Infrared serial data out	FIRI/SIRI	GPIO2[13]	1	O	14-1/14-3
IR1_RXD	Infrared serial data in	FIRI/SIRI	GPIO2[14]	1	I	14-1/14-3
IR_CLKIN	Infrared interface shared clock	FIRI/SIRI	—	1	I	14-1/14-3
IR2_TXD	Infrared serial data out	FIRI/SIRI	UART_SOUT1	1	O	14-1/14-3
IR2_RXD	Infrared serial data in	FIRI/SIRI	GPIO2[7]/UART_SIN1	1	I	14-1/14-3
SPI Signals						
SPIMOSI	Master output slave input	SPI	UART_SIN0/GPIO2[5]	1	I/O	15.3/15-6
SPIMISO	Master input slave output	SPI	UART_SOUT0	1	I/O	15.3/15-6
$\overline{\text{SPISEL}}$	SPI slave select	SPI	GPIO2[12]/IIC2_SDA	1	I	15.3/15-6
SPICLK	SPI clock	SPI	GPIO2[11]/IIC2_SCL	1	I	15.3/15-6
SSI Signals						
SSI1_RXD	SSI receive data 1	SSI	LA15	1	I	
SSI1_TXD	SSI transmit data 1	SSI	LA10	1	O	
SSI1_RFS	SSI receive frame sync 1	SSI	LA14	1	I/O	
SSI1_TFS	SSI transmit frame sync 1	SSI	LA11	1	I/O	
SSI1_RCK	SSI receive clock 1	SSI	LA13	1	I/O	
SSI1_TCK	SSI transmit clock 1	SSI	LA12	1	I/O	
SSI2_RXD	SSI receive data 2	SSI	LA21	1	I	
SSI2_TXD	SSI transmit data 2	SSI	LA16	1	O	
SSI2_RFS	SSI receive frame sync 2	SSI	LA20	1	I/O	

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
SSI2_TFS	SSI transmit frame sync 2	SSI	LA17	1	I/O	
SSI2_RCK	SSI receive clock 2	SSI	LA19	1	I/O	
SSI2_TCK	SSI transmit clock 2	SSI	LA18	1	I/O	
DMA Signals						
DMA1_DREQ0	DMA request 0	DMA	GPIO2[24]/IRQ6	1	I	19.2/19-4
DMA1_DREQ3	DMA request 3	DMA	GPIO2[26]/IRQ9	1	I	19.2/19-4
DMA1_DACK0	DMA acknowledge 0	DMA	GPIO2[25]/IRQ7	1	O	19.2/19-4
DMA1_DACK3	DMA acknowledge 3	DMA	GPIO2[27]/IRQ10	1	O	19.2/19-4
DMA1_DDONE0	DMA done 0	DMA	IRQ8	1	O	19.2/19-4
DMA1_DDONE3	DMA done 3	DMA	GPIO2[28]/IRQ11	1	O	19.2/19-4
DMA2_DREQ0	DMA request 0	DMA	LCS5	1	I	19.2/19-4
DMA2_DREQ3	DMA request 3	DMA	GPIO2[29]	1	I	19.2/19-4
DMA2_DACK0	DMA acknowledge 0	DMA	LCS6	1	O	19.2/19-4
DMA2_DACK3	DMA acknowledge 3	DMA	GPIO2[30]	1	O	19.2/19-4
DMA2_DDONE0	DMA done 0	DMA	LCS7	1	O	19.2/19-4
DMA2_DDONE3	DMA done 3	DMA	GPIO2[31]	1	O	19.2/19-4
General-Purpose Timer Signals						
GTM1_TIN1	Timer 1 capture control signal 1	Global Timers	GPIO2[15]	1	I	17-2/17-5
GTM1_TIN3	Timer 1 capture control signal 3	Global Timers	GPIO2[21]	1	I	17-2/17-5
GTM1_TGATE1	Timer 1 counter gate control signal 1	Global Timers	GPIO2[16]	1	I	17-2/17-5
GTM1_TGATE3	Timer 1 counter gate control signal 3	Global Timers	GPIO2[22]	1	I	17-2/17-5
GTM1_TOUT1	Timer 1 counter output signal 1	Global Timers	GPIO2[17]	1	O	17-2/17-5
GTM1_TOUT3	Timer 1 counter output signal 3	Global Timers	GPIO2[23]	1	O	17-2/17-5
GTM2_TIN1	Timer 2 capture control signal 1	Global Timers	GPIO2[18]	1	I	17-2/17-5
GTM2_TGATE1	Timer 2 counter gate control signal 1	Global Timers	GPIO2[19]	1	I	17-2/17-5
GTM2_TOUT1	Timer 2 counter output signal 1	Global Timers	GPIO2[20]	1	O	17-2/17-5
PCI Signals						
PCI_AD[31:0]	muxed address/data	PCI	—	32	I/O	20.2/20-5
PCI_C/BE[3:0]	Command/byte enable	PCI	—	4	I/O	20.2/20-5
PCI_PAR	Parity	PCI	—	1	I/O	20.2/20-5
PCI_FRAME	Frame	PCI	—	1	I/O	20.2/20-5

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
PCI_TRDY	Target ready	PCI	—	1	I/O	20.2/20-5
PCI_IRDY	Initiator ready	PCI	—	1	I/O	20.2/20-5
PCI_STOP	Stop	PCI	—	1	I/O	20.2/20-5
PCI_DEVSEL	Device select	PCI	—	1	I/O	20.2/20-5
PCI_IDSEL	Init device select	PCI	—	1	I	20.2/20-5
PCI_PERR	Parity error	PCI	—	1	I/O	20.2/20-5
PCI_SERR	System error	PCI	—	1	I/O	20.2/20-5
PCI_REQ0	Request	PCI	—	1	I/O	20.2/20-5
PCI_REQ1	Request	PCI	GPIO1[0]	1	I/O	20.2/20-5
PCI_REQ2	Request	PCI	GPIO1[2]	1	I/O	20.2/20-5
PCI_REQ3	Request	PCI	GPIO1[4]	1	I/O	20.2/20-5
PCI_REQ4	Request	PCI	GPIO1[6]	1	I/O	20.2/20-5
PCI_GNT0	Grant	PCI	—	1	I/O	20.2/20-5
PCI_GNT1	Grant	PCI	GPIO1[1]	1	I/O	20.2/20-5
PCI_GNT2	Grant	PCI	GPIO1[3]	1	I/O	20.2/20-5
PCI_GNT3	Grant	PCI	GPIO1[5]	1	I/O	20.2/20-5
PCI_GNT4	Grant	PCI	GPIO1[7]	1	I/O	20.2/20-5
PCI_CLK	PCI clock	PCI	—	1	I	20.2/20-5
High-Speed IO Interface 1 Signals (SerDes 1)						
SD1_TX[3:0]	SerDes1 transmit data	SerDes1	—	4	O	21.2/21-4
SD1_TX[3:0]	SerDes1 transmit data complement	SerDes1	—	4	O	21.2/21-4
SD1_RX[3:0]	SerDes1 receive data	SerDes1	—	4	I	21.2/21-4
SD1_RX[3:0]	SerDes1 receive data complement	SerDes1	—	4	I	21.2/21-4
SD1_REF_CLK	SerDes1 reference clock	SerDes1	—	1	I	21.2/21-4
SD1_REF_CLK	SerDes1 reference clock complement	SerDes1	—	1	I	21.2/21-4
SD1_PLL_TPD	SerDes1 PLL test point digital	SerDes1	—	1	O	
High-Speed IO Interface 2 Signals (SerDes 2)						
SD2_TX[7:0]	SerDes2 transmit data	SerDes2	—	8	O	21.2/21-4
SD2_TX[7:0]	SerDes2 transmit data complement	SerDes2	—	8	O	21.2/21-4
SD2_RX[7:0]	SerDes2 receive data	SerDes2	—	8	I	21.2/21-4
SD2_RX[7:0]	SerDes2 receive data complement	SerDes2	—	8	I	21.2/21-4

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
SD2_REF_CLK	SerDes2 reference clock	SerDes2	—	1	I	21.2/21-4
$\overline{\text{SD2\_REF\_CLK}}$	SerDes2 reference clock complement	SerDes2	—	1	I	21.2/21-4
SD2_PLL_TPD	SerDes2 PLL test point digital	SerDes2	—	1	O	
General Purpose I/O Signals[1]						
GPIO1[0]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ1}}$	1	I/O	22-3/22-3
GPIO1[1]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT1}}$	1	I/O	22-3/22-3
GPIO1[2]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ2}}$	1	I/O	22-3/22-3
GPIO1[3]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT2}}$	1	I/O	22-3/22-3
GPIO1[4]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ3}}$	1	I/O	22-3/22-3
GPIO1[5]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT3}}$	1	I/O	22-3/22-3
GPIO1[6]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ4}}$	1	I/O	22-3/22-3
GPIO1[7]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT4}}$	1	I/O	22-3/22-3
GPIO1[8:15]	General-purpose input/output	GPIO	DIU_LD[16:23]	8	I/O	22-3/22-3
GPIO1[16]	General-purpose input/output	GPIO	DIU_LD0/ cfg_elbc_ecc	1	O	22-3/22-3
GPIO1[17]	General-purpose input/output	GPIO	DIU_LD1	1	O	22-3/22-3
GPIO1[18]	General-purpose input/output	GPIO	DIU_LD2	1	O	22-3/22-3
GPIO1[19]	General-purpose input/output	GPIO	DIU_LD3	1	O	22-3/22-3
GPIO1[20]	General-purpose input/output	GPIO	DIU_LD4/cfg_core_pll3	1	O	22-3/22-3
GPIO1[21]	General-purpose input/output	GPIO	DIU_LD5/cfg_lx1_lockov	1	O	22-3/22-3
GPIO1[22]	General-purpose input/output	GPIO	DIU_LD6/cfg_lx2_lockov	1	O	22-3/22-3
GPIO1[23]	General-purpose input/output	GPIO	DIU_LD7/cfg_io_ports0	1	O	22-3/22-3
GPIO1[24]	General-purpose input/output	GPIO	DIU_LD8/cfg_io_ports1	1	O	22-3/22-3
GPIO1[25]	General-purpose input/output	GPIO	DIU_LD9/cfg_io_ports2	1	O	22-3/22-3
GPIO1[26]	General-purpose input/output	GPIO	DIU_LD10/cfg_sys_pll0	1	O	22-3/22-3
GPIO1[27]	General-purpose input/output	GPIO	DIU_LD11/ cfg_dram_type0	1	O	22-3/22-3
GPIO1[28]	General-purpose input/output	GPIO	DIU_LD12/ cfg_dram_type1	1	O	22-3/22-3
GPIO1[29]	General-purpose input/output	GPIO	DIU_LD13/cfg_rom_loc1	1	O	22-3/22-3
GPIO1[30]	General-purpose input/output	GPIO	DIU_LD14/cfg_rom_loc2	1	O	22-3/22-3
GPIO1[31]	General-purpose input/output	GPIO	DIU_LD15/cfg_rom_loc3	1	O	22-3/22-3

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
GPIO2[5]	General-purpose input/output	GPIO	SPIMOSI/UART_SIN0	1	I/O	22-3/22-3
GPIO2[6]	General-purpose input/output	GPIO	UART_CTS0/ UART_SIN2	1	I/O	22-3/22-3
GPIO2[7]	General-purpose input/output	GPIO	IR2_RXD/UART_SIN1	1	I/O	22-3/22-3
GPIO2[8]	General-purpose input/output	GPIO	UART_CTS1/ UART_SIN3	1	I/O	22-3/22-3
GPIO2[9]	General-purpose input/output	GPIO	IIC1_SCL	1	I/O	22-3/22-3
GPIO2[10]	General-purpose input/output	GPIO	IIC1_SDA	1	I/O	22-3/22-3
GPIO2[11]	General-purpose input/output	GPIO	IIC2_SCL/SPICLK	1	I	22-3/22-3
GPIO2[12]	General-purpose input/output	GPIO	IIC2_SDA/SPISEL	1	I/O	22-3/22-3
GPIO2[13]	General-purpose input/output	GPIO	IR1_TXD	1	I/O	22-3/22-3
GPIO2[14]	General-purpose input/output	GPIO	IR1_RXD	1	I/O	22-3/22-3
GPIO2[15]	General-purpose input/output	GPIO	GTM1_TIN1	1	I/O	22-3/22-3
GPIO2[16]	General-purpose input/output	GPIO	GTM1_TGATE1	1	I/O	22-3/22-3
GPIO2[17]	General-purpose input/output	GPIO	GTM1_TOUT1	1	I/O	22-3/22-3
GPIO2[18]	General-purpose input/output	GPIO	GTM2_TIN1	1	I/O	22-3/22-3
GPIO2[19]	General-purpose input/output	GPIO	GTM2_TGATE1	1	I/O	22-3/22-3
GPIO2[20]	General-purpose input/output	GPIO	GTM2_TOUT1	1	I/O	22-3/22-3
GPIO2[21]	General-purpose input/output	GPIO	GTM1_TIN3	1	I/O	22-3/22-3
GPIO2[22]	General-purpose input/output	GPIO	GTM1_TGATE3	1	I/O	22-3/22-3
GPIO2[23]	General-purpose input/output	GPIO	GTM1_TOUT3	1	I/O	22-3/22-3
GPIO2[24]	General-purpose input/output	GPIO	DMA1_DREQ0/IRQ6	1	I/O	22-3/22-3
GPIO2[25]	General-purpose input/output	GPIO	DMA1_DACK0/IRQ7	1	I/O	22-3/22-3
GPIO2[26]	General-purpose input/output	GPIO	DMA1_DREQ3/IRQ9	1	I/O	22-3/22-3
GPIO2[27]	General-purpose input/output	GPIO	DMA1_DACK3/IRQ10	1	I/O	22-3/22-3
GPIO2[28]	General-purpose input/output	GPIO	DMA1_DDONE3/IRQ11	1	I/O	22-3/22-3
GPIO2[29]	General-purpose input/output	GPIO	DMA2_DREQ3	1	I/O	22-3/22-3
GPIO2[30]	General-purpose input/output	GPIO	DMA2_DACK3	1	I/O	22-3/22-3
GPIO2[31]	General-purpose input/output	GPIO	DMA2_DDONE3	1	I/O	22-3/22-3
System Control Signals						
HRESET	Hard reset	System control	—	1	I	4-2/4-2
HRESET_REQ	Hard reset request	System control	—	1	O	4-2/4-2
SRESET	Soft reset	System control	—	1	I	4-2/4-2

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
$\overline{\text{CKSTP\_IN}}$	Checkstop in	System control	—	1	I	23-2/23-2
$\overline{\text{CKSTP\_OUT}}$	Checkstop out	System control	—	1	O	23-2/23-2
$\overline{\text{SMI}}$	System management interrupt	System control	—	1	I	23-2/23-2
Power Management Signals						
ASLEEP	Asleep	Power mgmt	cfg_core_speed	1	O	23-2/23-2
Debug Signals						
TRIG_IN	Watchpoint trigger in	Debug	—	1	I	
TRIG_OUT/READY /$\overline{\text{QUIESCE}}$	Watchpoint trigger out	Debug		1	O	4-2/4-2
MSRCID0	Memory debug source port ID 0	Debug	cfg_mem_debug	1	O	
MSRCID1	Memory debug source port ID 1	Debug	—	1	O	
MSRCID[2:4]	Memory debug source port ID 2–4	Debug	—	3	O	
MDVAL	Memory debug data valid	Debug		1	O	
CLK_OUT	Clock out	Debug	—	1	O	23-2/23-2
Test Signals						
$\overline{\text{LSSD\_MODE}}$	LSSD mode	Test	—	1	I	
TEST_MODE0	Test mode 0	Test	—	1	I	
TEST_MODE1	Test mode 1	Test	—	1	I	
JTAG Signals						
TCK	Test clock	JTAG	—	1	I	
TDI	Test data in	JTAG	—	1	I	
TDO	Test data out	JTAG	—	1	O	
TMS	Test mode select	JTAG	—	1	I	
$\overline{\text{TRST}}$	Test reset	JTAG	—	1	I	
Analog Signals						
SD1_IMP_CAL_TX	SerDes1 transmit impedance calibration	Analog	—	1	A	—[2]
SD1_IMP_CAL_RX	SerDes1 receive impedance calibration	Analog	—	1	A	—[2]
SD2_IMP_CAL_TX	SerDes2 transmit impedance calibration	Analog	—	1	A	—[2]
SD2_IMP_CAL_RX	SerDes2 receive impedance calibration	Analog	—	1	A	—[2]

Table 3-1. MPC8610 Signal Reference by Functional Block (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
SD1_PLL_TPA	SerDes1 PLL test point analog	Analog	—	1	A	—[2]
SD2_PLL_TPA	SerDes2 PLL test point analog	Analog	—	1	A	—[2]
TEMP_ANODE	Thermal diode	Analog	—	1	A	—[2]
TEMP_CATHODE	Thermal diode	Analog	—	1	A	—[2]

[1] Most GPIO signals can be configured independently to be inputs, outputs, or input/output.
[2] See the MPC8610 Integrated Processor Hardware Specifications for more information.

Table 3-2 provides the summary list of the signals grouped alphabetically.

Table 3-2. MPC8610 Signal Names Alphabetical Reference

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
ASLEEP	Asleep	Power mgmt	cfg_core_speed	1	O	23-2/23-2
$\overline{\text{CKSTP\_IN}}$	Checkstop in	System control	—	1	I	23-2/23-2
$\overline{\text{CKSTP\_OUT}}$	Checkstop out	System control	—	1	O	23-2/23-2
CLK_OUT	Clock out	Debug	—	1	O	23-2/23-2
DIU_CLK_OUT	DIU pixel clock	LCD	—	1	O	10.2/10-2
DIU_DE	DIU data enable	LCD	cfg_rom_loc0	1	O	10.2/10-2
DIU_HSYNC	DIU horizontal sync	LCD	cfg_pci_arb	1	O	10.2/10-2
DIU_LD[16:23]	DIU data (Red[7:0])	LCD	GPIO1[8:15]	8	O	10.2/10-2
DIU_LD0	DIU data (Blue0)	LCD	GPIO1[16]/ cfg_elbc_ecc	1	O	10.2/10-2
DIU_LD1	DIU data (Blue1)	LCD	GPIO1[17]	1	O	10.2/10-2
DIU_LD10	DIU data (Green2)	LCD	GPIO1[26]/ cfg_sys_pll0	1	O	10.2/10-2
DIU_LD11	DIU data (Green3)	LCD	GPIO1[27]/ cfg_dram_type0	1	O	10.2/10-2
DIU_LD12	DIU data (Green4)	LCD	GPIO1[28]/ cfg_dram_type1	1	O	10.2/10-2
DIU_LD13	DIU data (Green5)	LCD	GPIO1[29]/ cfg_rom_loc1	1	O	10.2/10-2
DIU_LD14	DIU data (Green6)	LCD	GPIO1[30]/ cfg_rom_loc2	1	O	10.2/10-2
DIU_LD15	DIU data (Green7)	LCD	GPIO1[31]/ cfg_rom_loc3	1	O	10.2/10-2
DIU_LD2	DIU data (Blue2)	LCD	GPIO1[18]	1	O	10.2/10-2
DIU_LD3	DIU data (Blue3)	LCD	GPIO1[19]	1	O	10.2/10-2

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
DIU_LD4	DIU data (Blue4)	LCD	GPIO1[20]/ cfg_core_pll3	1	O	10.2/10-2
DIU_LD5	DIU data (Blue5)	LCD	GPIO1[21]/ cfg_lx1_lockov	1	O	10.2/10-2
DIU_LD6	DIU data (Blue6)	LCD	GPIO1[22]/ cfg_lx2_lockov	1	O	10.2/10-2
DIU_LD7	DIU data (Blue7)	LCD	GPIO1[23]/ cfg_io_ports0	1	O	10.2/10-2
DIU_LD8	DIU data (Green0)	LCD	GPIO1[24]/ cfg_io_ports1	1	O	10.2/10-2
DIU_LD9	DIU data (Green1)	LCD	GPIO1[25]/ cfg_io_ports2	1	O	10.2/10-2
DIU_VSYNC	DIU vertical sync	LCD	cfg_pci_impd	1	O	10.2/10-2
$\overline{\text{DMA1\_DACK0}}$	DMA acknowledge 0	DMA	GPIO2[25]/$\overline{\text{IRQ7}}$	1	O	19.2/19-4
$\overline{\text{DMA1\_DACK3}}$	DMA acknowledge 3	DMA	GPIO2[27]/$\overline{\text{IRQ10}}$	1	O	19.2/19-4
$\overline{\text{DMA1\_DDONE0}}$	DMA done 0	DMA	$\overline{\text{IRQ8}}$	1	O	19.2/19-4
$\overline{\text{DMA1\_DDONE3}}$	DMA done 3	DMA	GPIO2[28]/$\overline{\text{IRQ11}}$	1	O	19.2/19-4
$\overline{\text{DMA1\_DREQ0}}$	DMA request 0	DMA	GPIO2[24]/$\overline{\text{IRQ6}}$	1	I	19.2/19-4
$\overline{\text{DMA1\_DREQ3}}$	DMA request 3	DMA	GPIO2[26]/$\overline{\text{IRQ9}}$	1	I	19.2/19-4
$\overline{\text{DMA2\_DACK0}}$	DMA acknowledge 0	DMA	$\overline{\text{LCS6}}$	1	O	19.2/19-4
$\overline{\text{DMA2\_DACK3}}$	DMA acknowledge 3	DMA	GPIO2[30]	1	O	19.2/19-4
$\overline{\text{DMA2\_DDONE0}}$	DMA done 0	DMA	$\overline{\text{LCS7}}$	1	O	19.2/19-4
$\overline{\text{DMA2\_DDONE3}}$	DMA done 3	DMA	GPIO2[31]	1	O	19.2/19-4
$\overline{\text{DMA2\_DREQ0}}$	DMA request 0	DMA	$\overline{\text{LCS5}}$	1	I	19.2/19-4
$\overline{\text{DMA2\_DREQ3}}$	DMA request 3	DMA	GPIO2[29]	1	I	19.2/19-4
GPIO1[0]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ1}}$	1	I/O	22-3/22-3
GPIO1[1]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT1}}$	1	I/O	22-3/22-3
GPIO1[2]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ2}}$	1	I/O	22-3/22-3
GPIO1[3]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT2}}$	1	I/O	22-3/22-3
GPIO1[4]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ3}}$	1	I/O	22-3/22-3
GPIO1[5]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT3}}$	1	I/O	22-3/22-3
GPIO1[6]	General-purpose input/output	GPIO	$\overline{\text{PCI\_REQ4}}$	1	I/O	22-3/22-3
GPIO1[7]	General-purpose input/output	GPIO	$\overline{\text{PCI\_GNT4}}$	1	I/O	22-3/22-3
GPIO1[8:15]	General-purpose input/output	GPIO	DIU_LD[16:23]	8	I/O	22-3/22-3

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
GPIO1[16]	General-purpose input/output	GPIO	DIU_LD0/ cfg_elbc_ecc	1	O	22-3/22-3
GPIO1[17]	General-purpose input/output	GPIO	DIU_LD1	1	O	22-3/22-3
GPIO1[18]	General-purpose input/output	GPIO	DIU_LD2	1	O	22-3/22-3
GPIO1[19]	General-purpose input/output	GPIO	DIU_LD3	1	O	22-3/22-3
GPIO1[20]	General-purpose input/output	GPIO	DIU_LD4/cfg_core_pll3	1	O	22-3/22-3
GPIO1[21]	General-purpose input/output	GPIO	DIU_LD5/cfg_lx1_lockov	1	O	22-3/22-3
GPIO1[22]	General-purpose input/output	GPIO	DIU_LD6/cfg_lx2_lockov	1	O	22-3/22-3
GPIO1[23]	General-purpose input/output	GPIO	DIU_LD7/cfg_io_ports0	1	O	22-3/22-3
GPIO1[24]	General-purpose input/output	GPIO	DIU_LD8/cfg_io_ports1	1	O	22-3/22-3
GPIO1[25]	General-purpose input/output	GPIO	DIU_LD9/cfg_io_ports2	1	O	22-3/22-3
GPIO1[26]	General-purpose input/output	GPIO	DIU_LD10/cfg_sys_pll0	1	O	22-3/22-3
GPIO1[27]	General-purpose input/output	GPIO	DIU_LD11/ cfg_dram_type0	1	O	22-3/22-3
GPIO1[28]	General-purpose input/output	GPIO	DIU_LD12/ cfg_dram_type1	1	O	22-3/22-3
GPIO1[29]	General-purpose input/output	GPIO	DIU_LD13/cfg_rom_loc1	1	O	22-3/22-3
GPIO1[30]	General-purpose input/output	GPIO	DIU_LD14/cfg_rom_loc2	1	O	22-3/22-3
GPIO1[31]	General-purpose input/output	GPIO	DIU_LD15/cfg_rom_loc3	1	O	22-3/22-3
GPIO2[5]	General-purpose input/output	GPIO	SPIMOSI/UART_SIN0	1	I/O	22-3/22-3
GPIO2[6]	General-purpose input/output	GPIO	$\overline{\text{UART\_CTS0}}$/ UART_SIN2	1	I/O	22-3/22-3
GPIO2[7]	General-purpose input/output	GPIO	IR2_RXD/ UART_SIN1	1	I/O	22-3/22-3
GPIO2[8]	General-purpose input/output	GPIO	$\overline{\text{UART\_CTS1}}$/ UART_SIN3	1	I/O	22-3/22-3
GPIO2[9]	General-purpose input/output	GPIO	IIC1_SCL	1	I/O	22-3/22-3
GPIO2[10]	General-purpose input/output	GPIO	IIC1_SDA	1	I/O	22-3/22-3
GPIO2[11]	General-purpose input/output	GPIO	IIC2_SCL/ SPICLK	1	I	22-3/22-3
GPIO2[12]	General-purpose input/output	GPIO	IIC2_SDA/ $\overline{\text{SPISEL}}$	1	I/O	22-3/22-3
GPIO2[13]	General-purpose input/output	GPIO	IR1_TXD	1	I/O	22-3/22-3
GPIO2[14]	General-purpose input/output	GPIO	IR1_RXD	1	I/O	22-3/22-3
GPIO2[15]	General-purpose input/output	GPIO	GTM1_TIN1	1	I/O	22-3/22-3

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
GPIO2[16]	General-purpose input/output	GPIO	GTM1_TGATE1	1	I/O	22-3/22-3
GPIO2[17]	General-purpose input/output	GPIO	GTM1_TOUT1	1	I/O	22-3/22-3
GPIO2[18]	General-purpose input/output	GPIO	GTM2_TIN1	1	I/O	22-3/22-3
GPIO2[19]	General-purpose input/output	GPIO	GTM2_TGATE1	1	I/O	22-3/22-3
GPIO2[20]	General-purpose input/output	GPIO	GTM2_TOUT1	1	I/O	22-3/22-3
GPIO2[21]	General-purpose input/output	GPIO	GTM1_TIN3	1	I/O	22-3/22-3
GPIO2[22]	General-purpose input/output	GPIO	GTM1_TGATE3	1	I/O	22-3/22-3
GPIO2[23]	General-purpose input/output	GPIO	GTM1_TOUT3	1	I/O	22-3/22-3
GPIO2[24]	General-purpose input/output	GPIO	DMA1_DREQ0/IRQ6	1	I/O	22-3/22-3
GPIO2[25]	General-purpose input/output	GPIO	DMA1_DACK0/IRQ7	1	I/O	22-3/22-3
GPIO2[26]	General-purpose input/output	GPIO	DMA1_DREQ3/IRQ9	1	I/O	22-3/22-3
GPIO2[27]	General-purpose input/output	GPIO	DMA1_DACK3/IRQ10	1	I/O	22-3/22-3
GPIO2[28]	General-purpose input/output	GPIO	DMA1_DDONE3/IRQ11	1	I/O	22-3/22-3
GPIO2[29]	General-purpose input/output	GPIO	DMA2_DREQ3	1	I/O	22-3/22-3
GPIO2[30]	General-purpose input/output	GPIO	DMA2_DACK3	1	I/O	22-3/22-3
GPIO2[31]	General-purpose input/output	GPIO	DMA2_DDONE3	1	I/O	22-3/22-3
GTM1_TGATE1	Timer 1 counter gate control signal 1	Global Timers	GPIO2[16]	1	I	17-2/17-5
GTM1_TGATE3	Timer 1 counter gate control signal 3	Global Timers	GPIO2[22]	1	I	17-2/17-5
GTM1_TIN1	Timer 1 capture control signal 1	Global Timers	GPIO2[15]	1	I	17-2/17-5
GTM1_TIN3	Timer 1 capture control signal 3	Global Timers	GPIO2[21]	1	I	17-2/17-5
GTM1_TOUT1	Timer 1 counter output signal 1	Global Timers	GPIO2[17]	1	O	17-2/17-5
GTM1_TOUT3	Timer 1 counter output signal 3	Global Timers	GPIO2[23]	1	O	17-2/17-5
GTM2_TGATE1	Timer 2 counter gate control signal 1	Global Timers	GPIO2[19]	1	I	17-2/17-5
GTM2_TIN1	Timer 2 capture control signal 1	Global Timers	GPIO2[18]	1	I	17-2/17-5
GTM2_TOUT1	Timer 2 counter output signal 1	Global Timers	GPIO2[20]	1	O	17-2/17-5
HRESET	Hard reset	System control	—	1	I	4-2/4-2
HRESET_REQ	Hard reset request	System control	—	1	O	4-2/4-2
IIC1_SCL	I^2C serial clock	I^2C	GPIO2[9]	1	I/O	12.3/12-3

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
IIC1_SDA	I^2C serial data	I^2C	GPIO2[10]	1	I/O	12.3/12-3
IIC2_SCL	I^2C serial clock	I^2C	GPIO2[11] /SPICLK	1	I/O	12.3/12-3
IIC2_SDA	I^2C serial data	I^2C	GPIO2[12]/$\overline{\text{SPISEL}}$	1	I/O	12.3/12-3
IR_CLKIN	Infrared interface shared clock	FIRI/SIRI	—	1	I	14-1/14-3
IR1_RXD	Infrared serial data in	FIRI/SIRI	GPIO2[14]	1	I	14-1/14-3
IR1_TXD	Infrared serial data out	FIRI/SIRI	GPIO2[13]	1	O	14-1/14-3
IR2_RXD	Infrared serial data in	FIRI/SIRI	GPIO2[7]/ UART_SIN1	1	I	14-1/14-3
IR2_TXD	Infrared serial data out	FIRI/SIRI	UART_SOUT1	1	O	14-1/14-3
IRQ[0:5]	External interrupt 0–5	PIC	—	6	I	11.2/11-7
$\overline{\text{IRQ\_OUT}}$	Interrupt output	PIC	—	1	O	11.2/11-7
IRQ10	External interrupt 10	PIC	$\overline{\text{DMA1\_DACK3}}$	1	I	11.2/11-7
IRQ11	External interrupt 11	PIC	$\overline{\text{DMA1\_DDONE3}}$	1	I	11.2/11-7
IRQ6	External interrupt 6	PIC	$\overline{\text{DMA1\_DREQ0}}$	1	I	11.2/11-7
IRQ7	External interrupt 7	PIC	$\overline{\text{DMA1\_DACK0}}$	1	I	11.2/11-7
IRQ8	External interrupt 8	PIC	$\overline{\text{DMA1\_DDONE0}}$	1	I	11.2/11-7
IRQ9	External interrupt 9	PIC	$\overline{\text{DMA1\_DREQ3}}$	1	I	11.2/11-7
LA[22:24]	Non-multiplexed address bus	eLBC	—	3	O	9.2/9-4
LA[28:31]	Non-multiplexed address bus	eLBC	cfg_sys_pll[1:4]	4	O	9.2/9-4
LA10	Non-multiplexed address bus	eLBC	SSI1_TXD/ cfg_ssi_la_sel	1	O	9.2/9-4
LA11	Non-multiplexed address bus	eLBC	SSI1_TFS	1	O	9.2/9-4
LA12	Non-multiplexed address bus	eLBC	SSI1_TCK	1	O	9.2/9-4
LA13	Non-multiplexed address bus	eLBC	SSI1_RCK	1	O	9.2/9-4
LA14	Non-multiplexed address bus	eLBC	SSI1_RFS	1	O	9.2/9-4
LA15	Non-multiplexed address bus	eLBC	SSI1_RXD	1	O	9.2/9-4
LA16	Non-multiplexed address bus	eLBC	SSI2_TXD	1	O	9.2/9-4
LA17	Non-multiplexed address bus	eLBC	SSI2_TFS	1	O	9.2/9-4
LA18	Non-multiplexed address bus	eLBC	SSI2_TCK	1	O	9.2/9-4
LA19	Non-multiplexed address bus	eLBC	SSI2_RCK	1	O	9.2/9-4
LA20	Non-multiplexed address bus	eLBC	SSI2_RFS	1	O	9.2/9-4
LA21	Non-multiplexed address bus	eLBC	SSI2_RXD	1	O	9.2/9-4

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
LA25	Non-multiplexed address bus	eLBC	cfg_elbc_clkdiv0	1	O	9.2/9-4
LA26	Non-multiplexed address bus	eLBC	cfg_elbc_clkdiv1	1	O	9.2/9-4
LA27	Non-multiplexed address bus	eLBC	cfg_cpu_boot	1	O	9.2/9-4
LAD[0:31]	muxed data/address	eLBC	cfg_gpinput[0:31]	32	I/O	9.2/9-4
LALE	Address latch enable	eLBC	cfg_core_pll1	1	O	9.2/9-4
LBCTL	Buffer control	eLBC	cfg_core_pll0	1	O	9.2/9-4
LCLK[0:2]	Local bus clocks	eLBC	—	3	O	9.2/9-4
$\overline{\text{LCS}}$[0:4]	Chip selects	eLBC	—	5	O	9.2/9-4
$\overline{\text{LCS5}}$	Chip Selects / DMA Triggers	eLBC	$\overline{\text{DMA2\_DREQ0}}$	1	O	9.2/9-4
$\overline{\text{LCS6}}$	Chip Selects / DMA Triggers	eLBC	$\overline{\text{DMA2\_DACK0}}$	1	O	9.2/9-4
$\overline{\text{LCS7}}$	Chip Selects / DMA Triggers	eLBC	$\overline{\text{DMA2\_DDONE0}}$	1	O	9.2/9-4
LDP[0:3]/LA[6:9]	Data parity/non-multiplexed address bus	eLBC	—	4	I/O	9.2/9-4
LGPL0/ LFCLE	UPM general-purpose line 0/ FCM flash command latch enable	eLBC	cfg_net2_div	1	O	9.2/9-4
LGPL1/ LFALE	UPM general-purpose line 1/ FCM flash address latch enable	eLBC	cfg_pci_clk	1	O	9.2/9-4
LGPL2/ $\overline{\text{LOE}}$/ $\overline{\text{LFRE}}$	UPM general-purpose line 2 / GPCM output enable/FCM flash read enable	eLBC	cfg_core_pll2	1	O	9.2/9-4
LGPL3/ $\overline{\text{LFWP}}$	UPM general-purpose line 3 / FCM flash write protect	eLBC	cfg_boot_seq0	1	O	9.2/9-4
LGPL5	UPM general-purpose line 5	eLBC	cfg_boot_seq1	1	O	9.2/9-4
$\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/ LGPL4/LUPWAIT/ LPBSE	GPCM target ack/ FCM flash ready(busy)/ UPM general-purpose line 4/ UPM wait/parity byte select	eLBC	—	1	I/O	9.2/9-4
$\overline{\text{LSSD\_MODE}}$	LSSD mode	Test	—	1	I	
$\overline{\text{LWE0}}$/$\overline{\text{LFWE}}$/ $\overline{\text{LBS0}}$	Write enable 0/LFWE/byte select 0	eLBC	cfg_pci_speed	1	O	9.2/9-4
$\overline{\text{LWE1}}$/$\overline{\text{LBS1}}$	Write enable 1/byte select 1	eLBC	cfg_host_agt0	1	O	9.2/9-4
$\overline{\text{LWE2}}$/$\overline{\text{LBS2}}$	Write enable 2/byte select 2	eLBC	cfg_host_agt1	1	O	9.2/9-4
$\overline{\text{LWE3}}$/$\overline{\text{LBS3}}$	Write enable 3/byte select 3	eLBC	cfg_host_agt2	1	O	9.2/9-4
MA[15:0]	Address bus	DDR Memory	—	16	O	8.3/8-3
MBA[2:0]	Logical bank address	DDR Memory	—	3	O	8.3/8-3

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
$\overline{\text{MCAS}}$	Column address strobe	DDR Memory	—	1	O	8.3/8-3
MCK[0:5]	DRAM clock outputs	DDR Memory	—	6	O	8.3/8-3
$\overline{\text{MCK}}$[0:5]	DRAM clock outputs (complement)	DDR Memory	—	6	O	8.3/8-3
MCKE[0:3]	DRAM clock enable	DDR Memory	—	4	O	8.3/8-3
$\overline{\text{MCP}}$	Machine check	PIC	—	1	I	
$\overline{\text{MCS}}$[0:3]	Chip selects	DDR Memory	—	4	O	8.3/8-3
MDIC[0:1]	Driver impedance calibration	DDR Memory	—	2	I/O	8.3/8-3
MDM[0:8]	Data mask	DDR Memory	—	9	O	8.3/8-3
MDQ[0:63]	Data bus	DDR Memory	—	64	I/O	8.3/8-3
MDQS[0:8]	Data strobes	DDR Memory	—	9	I/O	8.3/8-3
$\overline{\text{MDQS}}$[0:8]	Complement data strobes	DDR Memory	—	9	I/O	8.3/8-3
MDVAL	Memory debug data valid	Debug		1	O	
MECC[0:7]	Error checking and correcting	DDR Memory	—	8	I/O	8.3/8-3
MODT[0:3]	DRAM on-die termination	DDR Memory	—	4	O	8.3/8-3
$\overline{\text{MRAS}}$	Row address strobe	DDR Memory	—	1	O	8.3/8-3
MSRCID[2:4]	Memory debug source port ID 2–4	Debug	—	3	O	
MSRCID0	Memory debug source port ID 0	Debug	cfg_mem_debug	1	O	
MSRCID1	Memory debug source port ID 1	Debug	—	1	O	
$\overline{\text{MWE}}$	Write enable	DDR Memory	—	1	O	8.3/8-3
PCI_AD[31:0]	muxed address/data	PCI	—	32	I/O	20.2/20-5
PCI_C/$\overline{\text{BE}}$[3:0]	Command/byte enable	PCI	—	4	I/O	20.2/20-5
PCI_CLK	PCI clock	PCI	—	1	I	20.2/20-5
$\overline{\text{PCI\_DEVSEL}}$	Device select	PCI	—	1	I/O	20.2/20-5
$\overline{\text{PCI\_FRAME}}$	Frame	PCI	—	1	I/O	20.2/20-5
$\overline{\text{PCI\_GNT0}}$	Grant	PCI	—	1	I/O	20.2/20-5
$\overline{\text{PCI\_GNT1}}$	Grant	PCI	GPIO1[1]	1	I/O	20.2/20-5
$\overline{\text{PCI\_GNT2}}$	Grant	PCI	GPIO1[3]	1	I/O	20.2/20-5
$\overline{\text{PCI\_GNT3}}$	Grant	PCI	GPIO1[5]	1	I/O	20.2/20-5
$\overline{\text{PCI\_GNT4}}$	Grant	PCI	GPIO1[7]	1	I/O	20.2/20-5
PCI_IDSEL	Init device select	PCI	—	1	I	20.2/20-5

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
PCI_IRDY	Initiator ready	PCI	—	1	I/O	20.2/20-5
PCI_PAR	Parity	PCI	—	1	I/O	20.2/20-5
PCI_PERR	Parity error	PCI	—	1	I/O	20.2/20-5
PCI_REQ0	Request	PCI	—	1	I/O	20.2/20-5
PCI_REQ1	Request	PCI	GPIO1[0]	1	I/O	20.2/20-5
PCI_REQ2	Request	PCI	GPIO1[2]	1	I/O	20.2/20-5
PCI_REQ3	Request	PCI	GPIO1[4]	1	I/O	20.2/20-5
PCI_REQ4	Request	PCI	GPIO1[6]	1	I/O	20.2/20-5
PCI_SERR	System error	PCI	—	1	I/O	20.2/20-5
PCI_STOP	Stop	PCI	—	1	I/O	20.2/20-5
PCI_TRDY	Target ready	PCI	—	1	I/O	20.2/20-5
RTC	Real time clock	Clock	—	1	I	4-3/4-3
SD1_IMP_CAL_RX	SerDes1 receive impedance calibration	Analog	—	1	A	—[1]
SD1_IMP_CAL_TX	SerDes1 transmit impedance calibration	Analog	—	1	A	—[1]
SD1_PLL_TPA	SerDes1 PLL test point analog	Analog	—	1	A	—[1]
SD1_PLL_TPD	SerDes1 PLL test point digital	SerDes1	—	1	O	
SD1_REF_CLK	SerDes1 reference clock	SerDes1	—	1	I	21.2/21-4
SD1_REF_CLK	SerDes1 reference clock complement	SerDes1	—	1	I	21.2/21-4
SD1_RX[3:0]	SerDes1 receive data	SerDes1	—	4	I	21.2/21-4
SD1_RX[3:0]	SerDes1 receive data complement	SerDes1	—	4	I	21.2/21-4
SD1_TX[3:0]	SerDes1 transmit data	SerDes1	—	4	O	21.2/21-4
SD1_TX[3:0]	SerDes1 transmit data complement	SerDes1	—	4	O	21.2/21-4
SD2_IMP_CAL_RX	SerDes2 receive impedance calibration	Analog	—	1	A	—[1]
SD2_IMP_CAL_TX	SerDes2 transmit impedance calibration	Analog	—	1	A	—[1]
SD2_PLL_TPA	SerDes2 PLL test point analog	Analog	—	1	A	—[1]
SD2_PLL_TPD	SerDes2 PLL test point digital	SerDes2	—	1	O	
SD2_REF_CLK	SerDes2 reference clock	SerDes2	—	1	I	21.2/21-4

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
$\overline{\text{SD2\_REF\_CLK}}$	SerDes2 reference clock complement	SerDes2	—	1	I	21.2/21-4
SD2_RX[7:0]	SerDes2 receive data	SerDes2	—	8	I	21.2/21-4
$\overline{\text{SD2\_RX}}$[7:0]	SerDes2 receive data complement	SerDes2	—	8	I	21.2/21-4
SD2_TX[7:0]	SerDes2 transmit data	SerDes2	—	8	O	21.2/21-4
$\overline{\text{SD2\_TX}}$[7:0]	SerDes2 transmit data complement	SerDes2	—	8	O	21.2/21-4
$\overline{\text{SMI}}$	System management interrupt	System control	—	1	I	23-2/23-2
SPICLK	SPI clock	SPI	GPIO2[11]/IIC2_SCL	1	I	15.3/15-6
SPIMISO	Master input slave output	SPI	UART_SOUT0	1	I/O	15.3/15-6
SPIMOSI	Master output slave input	SPI	UART_SIN0/GPIO2[5]	1	I/O	15.3/15-6
$\overline{\text{SPISEL}}$	SPI slave select	SPI	GPIO2[12]/IIC2_SDA	1	I	15.3/15-6
$\overline{\text{SRESET}}$	Soft reset	System control	—	1	I	4-2/4-2
SSI1_RCK	SSI receive clock 1	SSI	LA13	1	I/O	
SSI1_RFS	SSI receive frame sync 1	SSI	LA14	1	I/O	
SSI1_RXD	SSI receive data 1	SSI	LA15	1	I	
SSI1_TCK	SSI transmit clock 1	SSI	LA12	1	I/O	
SSI1_TFS	SSI transmit frame sync 1	SSI	LA11	1	I/O	
SSI1_TXD	SSI transmit data 1	SSI	LA10	1	O	
SSI2_RCK	SSI receive clock 2	SSI	LA19	1	I/O	
SSI2_RFS	SSI receive frame sync 2	SSI	LA20	1	I/O	
SSI2_RXD	SSI receive data 2	SSI	LA21	1	I	
SSI2_TCK	SSI transmit clock 2	SSI	LA18	1	I/O	
SSI2_TFS	SSI transmit frame sync 2	SSI	LA17	1	I/O	
SSI2_TXD	SSI transmit data 2	SSI	LA16	1	O	
SYSCLK	System clock/PCI clock	Clock	—	1	I	4-3/4-3
TCK	Test clock	JTAG	—	1	I	
TDI	Test data in	JTAG	—	1	I	
TDO	Test data out	JTAG	—	1	O	
TEMP_ANODE	Thermal diode	Analog	—	1	A	—[1]
TEMP_CATHODE	Thermal diode	Analog	—	1	A	—[1]
TEST_MODE0	Test mode 0	Test	—	1	I	

Table 3-2. MPC8610 Signal Names Alphabetical Reference (continued)

Name	Description	Functional Block	Alternate Function(s)	No. of Signals	I/O	Table/Page
TEST_MODE1	Test mode 1	Test	—	1	I	
TMS	Test mode select	JTAG	—	1	I	
TRIG_IN	Watchpoint trigger in	Debug	—	1	I	
TRIG_OUT/READY/$\overline{Q}$ $\overline{UIESCE}$	Watchpoint trigger out	Debug		1	O	4-2/4-2
$\overline{TRST}$	Test reset	JTAG	—	1	I	
$\overline{UART\_CTS0}$	DUART clear to send	DUART 1	GPIO2[6]/ UART_SIN2	1	I	13.2/13-4
$\overline{UART\_CTS1}$	DUART clear to send	DUART 1	GPIO2[8]/ UART_SIN3	1	I	13.2/13-4
$\overline{UART\_RTS0}$	DUART ready to send	DUART 1	cfg_wdt_en/ UART_SOUT2	1	O	13.2/13-4
$\overline{UART\_RTS1}$	DUART ready to send	DUART 1	UART_SOUT3	1	O	13.2/13-4
UART_SIN0	DUART serial data in	DUART 1	SPIMOSI/ GPIO2[5]	1	I	13.2/13-4
UART_SIN1	DUART serial data in	DUART 1	IR2_RXD/ GPIO2[7]	1	I	13.2/13-4
UART_SIN2	DUART serial data in	DUART 2	GPIO2[6]/ $\overline{UART\_CTS0}$	1	I	13.2/13-4
UART_SIN3	DUART serial data in	DUART 2	GPIO2[8]/ $\overline{UART\_CTS1}$	1	I	13.2/13-4
UART_SOUT0	DUART serial data out	DUART 1	SPIMISO	1	O	13.2/13-4
UART_SOUT1	DUART serial data out	DUART 1	IR2_TXD	1	O	13.2/13-4
UART_SOUT2	DUART serial data out	DUART 2	cfg_wdt_en/ $\overline{UART\_RTS0}$	1	O	13.2/13-4
UART_SOUT3	DUART serial data out	DUART 2	$\overline{UART\_RTS1}$	1	O	13.2/13-4

[1] See the MPC8610 Integrated Processor Hardware Specifications for more information.

3.2 GPIO Signal Multiplexing

The general purpose I/O signals are extensively multiplexed with other functional signals on this device. This multiplexing is controlled by programmable fields in the general purpose I/O control register (GPIOCR) in the global utilities block. See Section 23.4.1.8, "General-Purpose I/O Control Register (GPIOCR)," for more information. The following sections describe the signaling options allowed by the GPIOCR. Note that, except for the I2C interfaces, the GPIOCR defaults to GPIO functionality on the respective signal pins; the GPIOCR must be initialized to the desired signaling for proper operation on the affected interface.

3.2.1 PCI Arbitration and GPIO1 Signal Multiplexing

The PCI bus $\overline{\text{REQ}}$/$\overline{\text{GNT}}$[1:4] signals are shared with GPIO1. The functionality of these signals is determined by the PCI_RG1, PCI_RG2, PCI_RG3, and PCI_RG4 fields in GPIOCR as shown in Table 3-3.

Table 3-3. PCI Arbitration Signal Configuration

Signal Name	Signal Function	GPIOCR Setting
$\overline{\text{PCI\_REQ1}}$	GPIO1[0]	PCI_RG1 = 0
	$\overline{\text{PCI\_REQ1}}$	PCI_RG1 = 1
$\overline{\text{PCI\_GNT1}}$	GPIO1[1]	PCI_RG1 = 0
	$\overline{\text{PCI\_GNT1}}$	PCI_RG1 = 1
$\overline{\text{PCI\_REQ2}}$	GPIO1[2]	PCI_RG2 = 0
	$\overline{\text{PCI\_REQ2}}$	PCI_RG2 = 1
$\overline{\text{PCI\_GNT2}}$	GPIO1[3]	PCI_RG2 = 0
	$\overline{\text{PCI\_GNT2}}$	PCI_RG2 = 1
$\overline{\text{PCI\_REQ3}}$	GPIO1[4]	PCI_RG3 = 0
	$\overline{\text{PCI\_REQ3}}$	PCI_RG3 = 1
$\overline{\text{PCI\_GNT3}}$	GPIO1[5]	PCI_RG3 = 0
	$\overline{\text{PCI\_GNT3}}$	PCI_RG3 = 1
$\overline{\text{PCI\_REQ4}}$	GPIO1[6]	PCI_RG4 = 0
	$\overline{\text{PCI\_REQ4}}$	PCI_RG4 = 1
$\overline{\text{PCI\_GNT4}}$	GPIO1[7]	PCI_RG4 = 0
	$\overline{\text{PCI\_GNT4}}$	PCI_RG4 = 1

3.2.2 Global Timer and GPIO2 Signal Multiplexing

The general purpose global timer signals are shared with GPIO2. The functionality of these signals is determined by the GTM1_1, GTM1_3, and GTM2_1 fields in GPIOCR as shown in Table 3-4.

Table 3-4. GTM Signal Configuration

Signal Name	Signal Function	GPIOCR Setting
GTM1_TIN1	GPIO2[15]	GTM1_1 = 0
	GTM1_TIN1	GTM1_1 = 1
GTM1_$\overline{\text{TGATE}}$1	GPIO2[16]	GTM1_1 = 0
	GTM1_$\overline{\text{TGATE}}$1	GTM1_1 = 1
GTM1_$\overline{\text{TOUT}}$1	GPIO2[17]	GTM1_1 = 0
	GTM1_$\overline{\text{TOUT}}$1	GTM1_1 = 1

Table 3-4. GTM Signal Configuration (continued)

Signal Name	Signal Function	GPIOCR Setting
GTM1_TIN3	GPIO2[21]	GTM1_3 = 0
	GTM1_TIN3	GTM1_3 = 1
GTM1_$\overline{\text{TGATE}}$3	GPIO2[22]	GTM1_3 = 0
	GTM1_$\overline{\text{TGATE}}$3	GTM1_3 = 1
GTM1_$\overline{\text{TOUT}}$3	GPIO2[23]	GTM1_3 = 0
	GTM1_$\overline{\text{TOUT}}$3	GTM1_3 = 1
GTM2_TIN1	GPIO2[18]	GTM2_1 = 0
	GTM2_TIN1	GTM2_1 = 1
GTM2_$\overline{\text{TGATE}}$1	GPIO2[19]	GTM2_1 = 0
	GTM2_$\overline{\text{TGATE}}$1	GTM2_1 = 1
GTM2_$\overline{\text{TOUT}}$1	GPIO2[20]	GTM2_1 = 0
	GTM2_$\overline{\text{TOUT}}$1	GTM2_1 = 1

3.2.3 DIU and GPIO1 Signal Multiplexing

The display interface unit (DIU) signals are shared with GPIO1. The functionality of these signals is determined by the DIU8 and DIU16 fields in GPIOCR as shown in Table 3-5.

Table 3-5. DIU Signal Configuration

Signal Name	Signal Function	GPIOCR Setting
DIU_LD[23:16]	GPIO1[15:8]	DIU8 = 0
	DIU_LD[23:16]	DIU8 = 1
DIU_LD[15:0]	GPIO1[31:16]	DIU16 = 0
	DIU_LD[15:0]	DIU16 = 1

3.2.4 IR1 and GPIO2 Signal Multiplexing

The fast infrared interface 1 (IR1) signals are shared with GPIO2. The functionality of these signals is determined by the IR1 field in GPIOCR as shown in Table 3-6.

Table 3-6. IR1 Signal Configuration

Signal Name	Signal Function	GPIOCR Setting
IR1_TXD	GPIO2[13]	IR1 = 0
	IR1_TXD	IR1 = 1
IR1_RXD	GPIO2[14]	IR1 = 0
	IR1_RXD	IR1 = 1

3.2.5 UART, SPI, and IR2, and GPIO2 Signal Multiplexing

For the UART signals, the following relationships apply:

- UART0 shares signals with GPIO2, SPI, and UART2
- UART1 shares signals with GPIO2, IR2, and UART3
- UART2 shares signals with GPIO2 and UART0
- UART3 shares signals with GPIO2 and UART1

The functionality of these signals is determined by the UART0, UART2, UART1, and UART3 fields in GPIOCR as shown in Table 3-7.

Table 3-7. UART Signal Configuration

Signal Name	Signal Function	GPIOCR Setting
UART_SIN0/SPIMOSI	GPIO2[5]	UART0 = 00
	SIN0	UART0 = 01
	SPIMOSI	UART0 = 11
UART_SOUT0/SPIMISO	SOUT0	UART0 = 01
	SPIMISO	UART0 = 11
$\overline{\text{UART\_CTS0}}$	GPIO2[6]	UART2 = 00
	$\overline{\text{CTS0}}$	UART2 = 01
	SIN2	UART2 = 11
$\overline{\text{UART\_RTS0}}$	$\overline{\text{RTS0}}$	UART2 = 01
	SOUT2	UART2 = 11
UART_SIN1/IR2_RXD	GPIO2[7]	UART1 = 00
	SIN1	UART1 = 01
	IR2_RXD	UART1 = 11
UART_SOUT1/IR2_TXD	SOUT1	UART1 = 01
	IR2_TXD	UART1 = 11
$\overline{\text{UART\_CTS1}}$	GPIO2[8]	UART3 = 00
	$\overline{\text{CTS1}}$	UART3 = 01
	SIN3	UART3 = 11
$\overline{\text{UART\_RTS1}}$	$\overline{\text{RTS1}}$	UART3 = 01
	SOUT3	UART3 = 11

3.2.6 I2C and GPIO2 Signal Multiplexing

The I^2C interface signals are shared with GPIO2. The functionality of the I2C1 signals is determined by the I2C1field in GPIOCR as shown in Table 3-8; the functionality of the I2C2 signals is determined by the UART0, I2C2_SCL and I2C2_SDA fields in GPIOCR also shown in Table 3-8.

Table 3-8. I2C Signal Configuration

Signal Name	Signal Function	GPIOCR Setting
IIC1_SCL	IIC1_SCL	I2C1 = 0
	GPIO2[9]	I2C1 = 1
IIC1_SDA	IIC1_SDA	I2C1 = 0
	GPIO2[10]	I2C1 = 1
IIC2_SCL/SPICLK	IIC2_SCL	UART0 ≠ 11 I2C2_SCL = 0
	GPIO2[11]	UART0 ≠ 11 I2C2_SCL = 1
	SPICLK	UART0 = 11
IIC2_SDA/$\overline{\text{SPISEL}}$	IIC2_SDA	UART0 ≠ 11 I2C2_SDA = 0
	$\overline{\text{SPISEL}}$	UART0 = 11 I2C2_SDA = 0
	GPIO2[12]	I2C2_SDA = 1

3.3 Configuration Signals Sampled at Reset

The signals that serve alternate functions as configuration input signals during system reset are summarized in Table 3-9. The detailed interpretation of their voltage levels during reset is described in Chapter 4, “Reset, Clocking, and Initialization.”

Note that throughout this document, the reset configuration signals are described as being sampled at the negation of $\overline{\text{HRESET}}$. However, there is a setup and hold time for these signals relative to the rising edge of $\overline{\text{HRESET}}$, as described in the MPC8610 *Integrated Processor Hardware Specifications*. Note that the PLL configuration signals have different setup and hold time requirements than the other reset configuration signals.

The reset configuration signals are multiplexed with other functional signals. The values on these signals during reset are interpreted to be logic one or zero, regardless of whether the functional signal name is defined as active-low. Most of the reset configuration signals have internal pull-up resistors so that if the signals are not driven, the default value is high (a one), as shown in the table. Some signals do not have pull-up resistors and must be driven high or low during the reset period. For details about all the signals that require external pull-up resistors, see the MPC8610 *Integrated Processor Hardware Specifications*.

Table 3-9. MPC8610 Reset Configuration Signals

Functional Block	Functional Signal Name	Reset Configuration Name	Default
eLBC	LGPL3/$\overline{\text{LFWP}}$	cfg_boot_seq0	1
eLBC	LGPL5	cfg_boot_seq1	1
eLBC	LBCTL	cfg_core_pll0	Must be driven
eLBC	LALE	cfg_core_pll1	Must be driven
eLBC	LGPL2/$\overline{\text{LOE}}$/$\overline{\text{LFRE}}$	cfg_core_pll2	Must be driven
eLBC	LA27]	cfg_cpu_boot	1
eLBC	LAD[0:31]	cfg_gpinput[0:31]	Indeterminate if not driven—no default
eLBC	$\overline{\text{LWE1}}$/$\overline{\text{LBS1}}$	cfg_host_agt0	1
eLBC	$\overline{\text{LWE2}}$/$\overline{\text{LBS2}}$	cfg_host_agt1	1
eLBC	$\overline{\text{LWE3}}$/$\overline{\text{LBS3}}$	cfg_host_agt2	1
eLBC	LGPL0/LFCLE	cfg_net2_div	1
eLBC	LGPL1/LFALE	cfg_pci_clk	1
eLBC	$\overline{\text{LWE0}}$/$\overline{\text{LFWE}}$/$\overline{\text{LBS0}}$	cfg_pci_speed	1
eLBC	LA10	cfg_ssi_la_sel	1
eLBC	LA25	cfg_elbc_clkdiv0	1
eLBC	LA26	cfg_elbc_clkdiv1	1
eLBC	LA[28:31]	cfg_sys_pll[1:4]	Must be driven
Debug	MSRCID0	cfg_mem_debug	1
Debug	MDVAL	cfg_boot_vector	1
Power mgmt	ASLEEP	cfg_core_speed	1
LCD	DIU_LD0/ GPIO[16]	cfg_elbc_ecc	1
LCD	DIU_LD4	cfg_core_pll3	Must be driven
LCD	DIU_LD5	cfg_lx1_lockov	1
LCD	DIU_LD6	cfg_lx2_lockov	1
LCD	DIU_LD7	cfg_io_ports0	1
LCD	DIU_LD8	cfg_io_ports1	1
LCD	DIU_LD9	cfg_io_ports2	1
LCD	DIU_LD10	cfg_sys_pll0	Must be driven
LCD	DIU_LD11	cfg_dram_type0	1
LCD	DIU_LD12	cfg_dram_type1	1
LCD	DIU_LD13	cfg_rom_loc1	1

Table 3-9. MPC8610 Reset Configuration Signals (continued)

Functional Block	Functional Signal Name	Reset Configuration Name	Default
LCD	DIU_LD14	cfg_rom_loc2	1
LCD	DIU_LD15	cfg_rom_loc3	1
LCD	DIU_VSYNC	cfg_pci_impd	1
LCD	DIU_HSYNC	cfg_pci_arb	1
LCD	DIU_DE	cfg_rom_loc0	1

3.4 Output Signal States During Reset

When a system reset is recognized ($\overline{\text{HRESET}}$ is asserted), the MPC8610 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 MPC8610 ignores most input signals (except for the reset configuration signals) and drives most of the output-only signals to an inactive state. Table 3-10 shows the states of the output-only signals (that is, the signals that are not multiplexed with other inputs, or are not used as reset configuration signals during system reset).

Table 3-10. Output Signal States During System Reset

Interface	Signal	State During Reset
DDR/DDR2	MA[15:0]	
DDR/DDR2	MBA[2:0]	
DDR/DDR2	$\overline{\text{MCS}}$[0:3]	
DDR/DDR2	MDM[0:8]	
DDR/DDR2	$\overline{\text{MCAS}}$	
DDR/DDR2	$\overline{\text{MWE}}$	
DDR/DDR2	$\overline{\text{MRAS}}$	
DDR/DDR2	MCK[0:5]	driven
DDR/DDR2	$\overline{\text{MCK}}$[0:5]	driven
DDR/DDR2	MCKE[0:3]	driven low
DDR/DDR2	MODT[0:3]	driven
eLBC	LA[10:24]	
eLBC	$\overline{\text{LCS}}$[0:7]	
eLBC	LCLK[0:2]	
LCD	DIU_LD[0:3]	

Table 3-10. Output Signal States During System Reset (continued)

Interface	Signal	State During Reset
LCD	DIU_CLK_OUT	
PIC	IRQ_OUT	
FIRI/SIRI	IR1_TXD	
FIRI/SIRI	IR2_TXD	
SSI	SSI1_TXD	
SSI	SSI2_TXD	
DMA	DMA1_DACK0	
DMA	DMA1_DACK3	
DMA	DMA1_DDONE0	
DMA	DMA1_DDONE3	
DMA	DMA2_DACK0	
DMA	DMA2_DACK3	
DMA	DMA2_DDONE0	
DMA	DMA2_DDONE3	
Global Timers	GTM1_TOUT1	
Global Timers	GTM1_TOUT3	
Global Timers	GTM2_TOUT1	
SerDes1	SD1_TX[3:0]	
SerDes1	SD1_TX[3:0]	
SerDes2	SD2_TX[7:0]	
SerDes2	SD2_TX[7:0]	
SerDes1	SD1_PLL_TPD	
SerDes2	SD2_PLL_TPD	
Debug	CLK_OUT	
JTAG	TDO	driven
System control	HRESET_REQ	—
System control	CKSTP_OUT	—

Chapter 4
Reset, Clocking, and Initialization

This chapter describes the reset, clocking, and some overall initialization of the MPC8610, including a definition of the reset configuration signals and the options they select. Additionally, the configuration, control, and status registers are described. Note that other chapters in this book may describe specific aspects of initialization for individual blocks.

4.1 Overview

The reset, clocking, and control signals provide many options for the operation of the device. Additionally, many modes are selected with reset configuration signals during the assertion of a hard reset (assertion of $\overline{\text{HRESET}}$).

4.2 External Signal Descriptions

Table 4-1 summarizes the external signals described in this chapter. Table 4-2 and Table 4-3 have detailed signal descriptions, but Table 4-1 contains references to additional sections that contain more information.

Table 4-1. Signal Summary

Signal	I/O	Description	References (Section/Page)
$\overline{\text{HRESET}}$	I	Hard reset input. Causes a power-on reset (POR) sequence.	4.4.1.2/4-8
$\overline{\text{HRESET\_REQ}}$	O	Hard reset request output. An internal block requests that $\overline{\text{HRESET}}$ be asserted	—
$\overline{\text{SRESET}}$	I	Soft reset input. Causes e600 core to service a soft reset exception.	4.4.1.1/4-8
READY (/TRIG_OUT)	O	The READY signal is functional when TOSR[SEL] = 0b000; this signal indicates that the device has completed the reset operation and is not in a power-down (nap, doze or sleep), checkstop, or debug state. See Chapter 25, "Debug Features and Watchpoint Facilities," for more information on TOSR and TRIG_OUT.	4.4.2/4-9
SYSCLK	I	Primary clock input to the device.	4.4.4.1/4-21
RTC	I	Real time clock input.	—

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

4.2.1 System Control Signals

Table 4-2 describes some of the system control signals of the device. Section 4.4.3, "Power-On Reset Configuration," describes the signals that also function as reset configuration signals. Note that the $\overline{\text{CKSTP\_IN}}$ and $\overline{\text{CKSTP\_OUT}}$ signals are described in Chapter 23, "Global Utilities."

Table 4-2. System Control Signals—Detailed Signal Descriptions

Signal	I/O	Description	
$\overline{\text{HRESET}}$	I	Hard reset. Causes the device to abort all current internal and external transactions and set all registers to their default values. $\overline{\text{HRESET}}$ may be asserted completely asynchronously with respect to all other signals.	
		State Meaning	Asserted/Negated—See Chapter 3, "Signal Descriptions," and Section 4.4.3, "Power-On Reset Configuration," for more information on the interpretation of the other device signals during reset.
		Timing	Assertion/Negation—The *MPC8610 Integrated Host Processor Hardware Specifications* gives specific timing information for this signal and the reset configuration signals.
$\overline{\text{HRESET\_REQ}}$	O	Hard reset request. Indicates to the board (system in which the device is embedded) that a condition requiring the assertion of $\overline{\text{HRESET}}$ has been detected.	
		State Meaning	Asserted—One of the following conditions has triggered a request for hard reset: • A boot sequencer failure (see Section 12.5.7, "Boot Sequencer Mode") • A watchdog timer reset (see description of SWCRR[SWRI] in Section 18.2.1, "System Watchdog Control Register (SWCRR)") Negated—Indicates no reset request.
		Timing	Assertion/Negation—May occur anytime, synchronous to the platform clock. Once asserted, $\overline{\text{HRESET\_REQ}}$ does not negate until $\overline{\text{HRESET}}$ is asserted.
$\overline{\text{SRESET}}$	I	Soft reset. An edge sensitive signal that causes a soft reset exception on the e600 core.	
		State Meaning	Asserted—Asserting $\overline{\text{SRESET}}$ causes a soft reset interrupt (edge sensitive) to e600 core. $\overline{\text{SRESET}}$ has no effect while $\overline{\text{HRESET}}$ is asserted. However, the POR sequence is paused if $\overline{\text{SRESET}}$ is asserted during POR.
		Timing	Assertion—May occur at any time, asynchronous to any clock. Negation—Must be asserted for at least two platform clock cycles.
READY (/TRIG_OUT)	O	Ready. Reset status signal.	
		State Meaning	Asserted—Indicates that the device has completed the reset operation and is not in a power-down state (nap, doze or sleep) when TOSR[SEL] equals 0b000. See Section 4.4.2, "Power-On Reset Sequence," for more information.
		Timing	Assertion/Negation—Initial assertion of READY after reset has been synchronized with SYSCLK. Subsequent assertion/negation due to power down modes occurs asynchronously.

4.2.2 Clock Signals

Table 4-3 describes the overall clock signals of the device. Note that some clock signals are specific to blocks within the device, and although some of their functionality is described in Section 4.4.4, "Clocking," they are defined in detail in their respective chapters.

Note that there is also a CLK_OUT signal in the device; the signal driven on the CLK_OUT pin is selectable and described in Section 23.4.1.25, "CLK_OUT Control Register (CLKOCR)."

Table 4-3. Clock Signals—Detailed Signal Descriptions

Signal	I/O	Description	
SYSCLK	I	System clock. SYSCLK is the primary clock input to the device. It is multiplied up with a platform phased-lock loop (PLL) based on cfg_sys_pll[0:4] at reset to create the internal platform clock (also called the MPX clock) which is used by virtually all of the synchronous system logic, including the DDR SDRAM and enhanced local bus controllers, and other internal blocks such as the DMA and interrupt controllers. The platform clock, in turn, feeds the e600 core PLL which creates the core clock based on cfg_core_pll[0:3] at reset.	
		Timing	Assertion/Negation—See the *MPC8610 Integrated Host Processor Hardware Specifications* for specific timing information for this signal.
RTC	I	Real time clock. This signal can be used to clock the global timers in the programmable interrupt controller (PIC).	
		Timing	Assertion/Negation—See the *MPC8610 Integrated Host Processor Hardware Specifications* for specific timing information for this signal.

4.3 Memory Map/Register Definition

There are no unique registers in the reset and clocking blocks of the device. However, this section describes the local configuration control registers; it also contains a brief description of the boot sequencer.

4.3.1 Local Configuration Control

The local configuration control registers are accessed by reading and writing to an address comprised of the configuration, control, and status base address (specified in the CCSRBAR), plus the local configuration control block base address (0x0_0000), plus the offset of the specific local configuration control register to be accessed. Note that the CCSRBAR is contained within the local configuration control register block and is thus self-referencing.

Table 4-4 shows the memory map for the local configuration control registers.

Table 4-4. Local Configuration Control Register Map

Offset (Hex)	Register	Access	Reset	Section/Page
Local Configuration Control Registers—Block Base Address 0x0_0000				
0x000	CCSRBAR—Configuration, control, and status registers base address register	R/W	0x000F_F700	4.3.1.1.3/4-4
0x008	ALTCBAR—Alternate configuration base address register	R/W	All zeros	4.3.1.2.1/4-5
0x010	ALTCAR—Alternate configuration attribute register	R/W	All zeros	4.3.1.2.2/4-6
0x020	BPTR—Boot page translation register	R/W	All zeros	4.3.1.3.1/4-7

4.3.1.1 Accessing Configuration, Control, and Status Registers

The configuration, control, and status registers (CCSRs) of the MPC8610 are memory mapped. These registers occupy a 1-Mbyte region of memory. Their location is programmable using the CCSR base address register (CCSRBAR). The default base address for the CCSRs is 0x0_FF70_0000 (CCSRBAR = 0x0_000F_F700).

CCSRBAR itself is part of the local configuration control block of CCSR memory. Because CCSRBAR is at offset 0x000 from the local configuration control block base address, CCSRBAR always points to itself.

4.3.1.1.1 POR I/O Impedance Control Register (PORIMPCR)

This register needs to get set early in the configuration data process. See Section 23.4.1.3, "POR I/O Impedance Control Register (PORIMPCR)," for details.

4.3.1.1.2 Updating CCSRBAR

Updates to CCSRBAR that relocate the entire 1-Mbyte region of CCSRs 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 are advised:

- The CCSRBAR should be updated during initial configuration of the device when only one host or controller has access to the device.
  - If the boot sequencer is being used to initialize the device, it is recommended that the boot sequencer set CCSRBAR to its desired final location.
  - If an external host on one of the PCI Express ports or PCI bus is configuring the device, it should set CCSRBAR to the desired final location before the e600 core is released to boot.
  - If the e600 core is initializing the device, it should set CCSRBAR to the desired final location before enabling other I/O devices to access the device.
- When the core is writing to CCSRBAR, it should use the following sequence:
  - The application program updating the core should make sure that other potential masters will not access the CCSRBAR register or configuration space pointed to by that register until the update is complete.
  - Read the current value of CCSRBAR using a load word instruction followed by an **isync**. This forces all accesses to configuration space to complete.
  - Write the new value to CCSRBAR.
  - Perform a load of an address that does not access configuration space but that has an address mapping already in effect (for example, boot ROM). Follow this load with an **isync**.
  - Read the contents of CCSRBAR from its new location, followed by another **isync**.

4.3.1.1.3 Configuration, Control, and Status Base Address Register (CCSRBAR)

Figure 4-1 shows the fields of CCSRBAR.

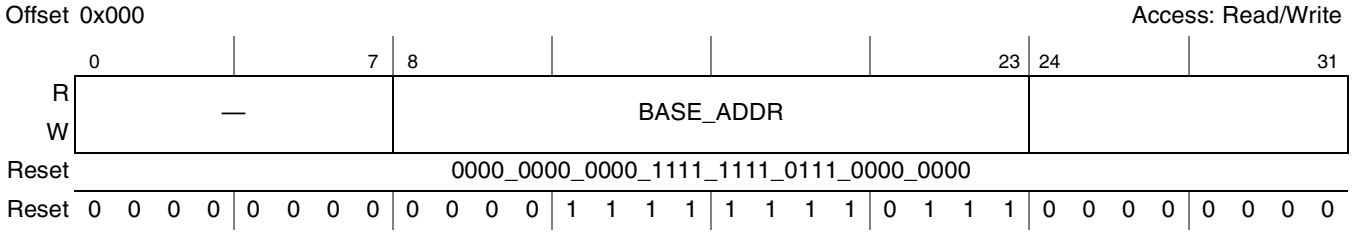


Figure 4-1. Configuration Control and Status Register Base Address Register (CCSRBAR)

Table 4-5 defines the bit fields of CCSRBAR.

Table 4-5. CCSRBAR Bit Settings

Bits	Name	Description
0–7	—	Write reserved, read = 0.
8–23	BASE_ADDR	Defines the16 most-significant address bits of the window used for configuration accesses. The base address is aligned on a 1-Mbyte boundary.
24–31	—	Write reserved, read = 0

4.3.1.2 Accessing Alternate Configuration Space

An alternate configuration space can be accessed by configuring the ALTCBAR and ALTCAR registers located in the MPX coherency module (MCM). These are intended to be used with the boot sequencer to allow the boot sequencer to access an alternate 1-Mbyte region of configuration space. 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 36-bit address that is mapped to the target specified in ALTCAR. Thus, by configuring these registers, the boot sequencer has access to the entire memory map, one 1-Mbyte block at a time. See Section 12.5.7, “Boot Sequencer Mode” for more information.

NOTE

The enable bit in the ALTCAR register should be cleared either by the boot sequencer or by the boot code that executes after the boot sequencer has completed its configuration operations. This prevents problems with incorrect mappings if subsequent configuration of the local access windows uses a different target mapping for the address specified in ALTCBAR.

4.3.1.2.1 Alternate Configuration Base Address Register (ALTCBAR)

Figure 4-2 shows the fields of ALTCBAR.

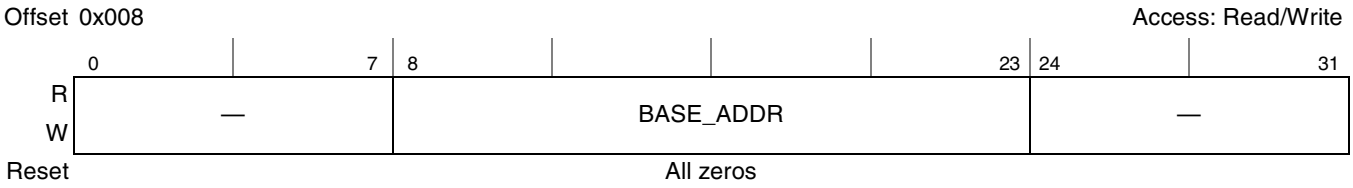


Figure 4-2. Alternate Configuration Base Address Register (ALTCBAR)

Table 4-6 defines the bit fields of ALTCBAR.

Table 4-6. ALTCBAR Bit Settings

Bits	Name	Description
0–7	—	Write reserved, read = 0
8–23	BASE_ADDR	Defines the 16 most significant address bits of an alternate window used for configuration accesses.
24–31	—	Write reserved, read = 0

4.3.1.2.2 Alternate Configuration Attribute Register (ALTCAR)

Figure 4-3 shows the fields of ALTCAR.

Figure 4-3. Alternate Configuration Attribute Register (ALTCAR)

Table 4-7 defines ALTCAR fields.

Table 4-7. ALTCAR Bit Settings

Bits	Name	Description
0	EN	Enable bit for a second configuration window. Like CCSRBAR, it has a fixed size of 1 Mbyte. 0 Second configuration window is disabled. 1 Second configuration window is enabled.
1–6	—	Write reserved, read = 0
7–11	TRGT	Identifies the target interface when a transaction hits in the 1-Mbyte address range defined by the alternate configuration window. The encodings for TRGT are defined in Table 2-4. 00000 PCI 00001 PCI Express 2 (×8) 00010 PCI Express 1 (×4) 00011 Reserved 00100 Local bus (eLBC) 00101–01110 Reserved 01111 DDR memory controller 10000–11111 Reserved
12–31	—	Write reserved, read = 0

4.3.1.3 Boot Page Translation

When the e600 core comes out of reset, it begins execution with the instruction at effective address 0x0_FFF0_0100. That is, the first instruction fetch is a burst read of boot code from effective address 0x0_FFF0_0100. For systems in which the boot code resides at a different address, the device provides boot page translation capability. Boot page translation is controlled by the boot page translation register (BPTR).

By default, the MCM always reserves a special boot window that covers the 8-Mbyte address range from 0x0_FF80_0000 to 0x0_FFFF_FFFF. This allows the core to issue their first instruction fetch without needing a local access window to be set up in the MCM. An address hit in the special boot window (the default window) is routed to the target defined by the values of the cfg_rom_loc[0:3] configuration signals defined in Section 4.4.3.8, "Boot ROM Location."

As an alternative, the BPTR can be used to identify the upper 16 most-significant address bits. These bits are used to translate an MPX bus initiated access to the last 4 Kbytes of the first 64-Gbytes of local address space. The translation is from an address at 0x0_FFF0_0xxx (the 4KB boot page) to an address at 0xy_yyy0_0xxx, where yyyy is BPTR[8:23]. If a boot controller device is used, the boot controller may write to BPTR (with the enable bit set) prior to the core issuing its first instruction fetch, thereby forcing the core to get the data from the translated address. This of course may require the boot sequencer to set up a local access window in the MCM that will route the translated boot address to the proper target. See Section 2.2, "Local Access Windows," and Section 12.5.7, "Boot Sequencer Mode," for more information.

The boot sequencer can enable boot page translation, or the boot page translation can be set up by an external host when the device is configured to be in boot holdoff mode. If translation is to be performed to a page outside the default boot ROM address range (0x0_FF80_0000 to 0x0_FFFF_FFFF), the external host or boot sequencer must then also set up a local access window to define the routing of the boot code fetch to the target interface that contains the boot code, because the BPTR defines only the address translation, not the target interface. See Section 2.2, "Local Access Windows," and Section 12.5.7, "Boot Sequencer Mode," for more information.

4.3.1.3.1 Boot Page Translation Register (BPTR)

Figure 4-4 shows the fields of BPTR.

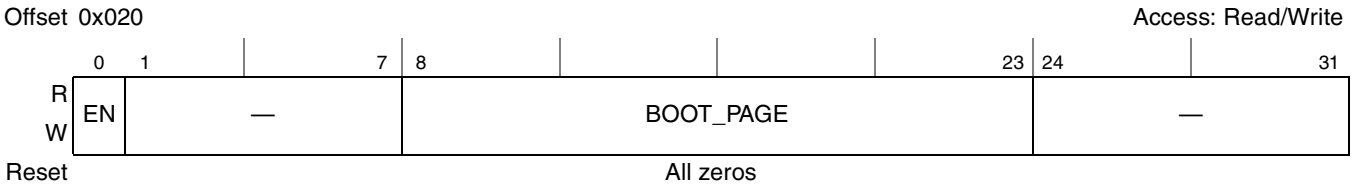


Figure 4-4. Boot Page Translation Register (BPTR)

Table 4-8 describes BPTR bit settings.

Table 4-8. BPTR Bit Settings

Bits	Name	Description
0	EN	Boot page translation enable. If set, the upper 16 address bits of any MPX bus initiated access to local address space from 4 Gb-1Mb to 16 Gbytes (0x0_FFxx_xxxx) is translated to the value in BOOT_PAGE. Note that the lower 8 bits of the vector are not translated and bits 16–31 are zero. 0 Boot page is not translated. 1 Boot page is translated as defined in the BPTR[BOOT_PAGE] parameter.
1–7	—	Write reserved, read = 0

Table 4-8. BPTR Bit Settings (continued)

Bits	Name	Description
8–23	BOOT_PAGE	Translation for boot page. If enabled, the high order 16 bits of address for accesses to 0x0_FFxx_xxxx are replaced with this value.
23–31	—	Write reserved, read = 0

4.3.2 Boot Sequencer

The boot sequencer is a specialized DMA engine that accesses a serial ROM on the primary I^2C interface and writes data to CCSR memory or the memory space pointed to by the alternate configuration base address register (ALTCBAR). See Section 4.3.1.1, “Accessing Configuration, Control, and Status Registers,” and Section 4.3.1.2, “Accessing Alternate Configuration Space.” The boot sequencer is enabled by reset configuration pins as described in Section 4.4.3.7, “Boot Sequencer Configuration.” If the boot sequencer is enabled, the e600 core is held in reset until the boot sequencer has completed its operation. For more details, see Section 12.5.7, “Boot Sequencer Mode” in the I^2C chapter. Note that the boot sequencer only works in conjunction with I^2C port 1.

4.4 Functional Description

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

4.4.1 Reset Operations

This section describes the reset input signals for hard and soft reset operation.

4.4.1.1 Soft Reset

Assertion of the $\overline{\text{SRESET}}$ signal causes a soft reset interrupt to the e600 core. This signal may be asserted asynchronously. It is a non-maskable exception and is recoverable. A soft reset interrupt is third in priority behind $\overline{\text{HRESET}}$ and a machine check.

4.4.1.2 Hard Reset

The device is completely reset by the assertion of the $\overline{\text{HRESET}}$ input. The assertion of this signal by external logic is the equivalent of a POR operation and causes the sequence of events described in Section 4.4.2, “Power-On Reset Sequence.”

Refer to the *MPC8610 Integrated Host Processor Hardware Specifications* for the timing requirements for $\overline{\text{HRESET}}$ assertion and negation.

The hard reset request output signal ($\overline{\text{HRESET\_REQ}}$) indicates to external logic that a hard reset is being requested by a boot sequencer failure (see Section 12.5.7, “Boot Sequencer Mode” and Section 12.5.7.5, “Boot Sequencer EEPROM Data Format”).

4.4.2 Power-On Reset Sequence

The POR sequence is as follows:

1. Power is applied to meet the specifications in the *MPC8610 Integrated Host Processor Hardware Specifications.*
2. System asserts $\overline{\text{HRESET}}$ and $\overline{\text{TRST}}$ causing all registers to be initialized to their default states and and releases all bidirectional I/O signals to a high-impedance state.
3. System applies a stable SYSCLK signal and stable PLL configuration inputs (cfg_sys_pll[0:3]), and the platform PLL begins locking to SYSCLK.
4. $\overline{\text{HRESET}}$ negates after a minimum of 100 µs. Note that POR configuration inputs should be valid prior to $\overline{\text{HRESET}}$ negation and held valid at least 4 SYSCLK cycles after $\overline{\text{HRESET}}$ has been negated.

NOTE

If the JTAG interface and COP header are not being used, Freescale recommends that $\overline{\text{TRST}}$ be tied to $\overline{\text{HRESET}}$ so that $\overline{\text{TRST}}$ is asserted when the $\overline{\text{HRESET}}$ is asserted, ensuring that the JTAG scan chain is initialized during the power-on reset flow. See the JTAG Configuration Signals section in *MPC8610 Integrated Host Processor Hardware Specifications* for more information.

There is no need to assert the $\overline{\text{SRESET}}$ signal when $\overline{\text{HRESET}}$ is asserted. If $\overline{\text{SRESET}}$ is asserted upon negation of $\overline{\text{HRESET}}$, the POR sequence will be paused until $\overline{\text{SRESET}}$ is negated.

5. The device enables the I/O drivers.
6. The e600 PLL configuration inputs (cfg_core_pll[0:3]) are applied, allowing the e600 core PLL to begin locking to the platform clock (*plat_clk*).
7. Once the *plat_clk* and cfg_core_pll[0:3] are stable, the time required to lock the e600 core PLL is approximately 100 µs plus 255 *plat_clk* cycles. The *core_clk* is then generated.
8. The internal hard reset to the e600 core is negated and soft reset is negated to the PLLs and other remaining I/O blocks. The PLLs begin to lock.
9. When PLL locking is completed, loading of configuration data can begin and complete from the I^2C if boot sequencing is enabled or directly from the boot ROM location determined by the cfg_rom_loc[0:3] (local bus, and others). See Section 4.4.3.7, “Boot Sequencer Configuration,” for details. Boot sequencing is disabled by default.
10. Booting the e600 core is allowed to proceed from the default boot vector fetch location unless boot hold off mode is enabled. If boot page translation is enabled in software, the fetch vector is based on the address programmed in the Boot Page Translation register, BPTR[Boot_Page] field, if BPTR[EN] is set; this could be done either in boot hold off mode or during boot sequencing. See Section 4.4.3.6, “CPU Boot Configuration,” for details. Alternatively, an alternate boot vector can use used if $\overline{\text{cfg\_alt\_boot\_vec}}$ is enabled during POR. The device is now in its ready state.

11. The ASLEEP signal negates synchronized to a rising edge of SYSCLK, indicating the ready state. The ready state is also indicated by the assertion of READY/TRIG_OUT, if TOSR[SEL] = 000. In this case, READY is asserted with the same rising edge of SYSCLK, to indicate that the device has reached its ready state.

    Asserting READYallows external system monitors to know basic device status. For example, it indicates exactly when it emerges from reset, or if the device is in a low-power mode. For more information on the debug functions of TRIG_OUT, see Section 25.3.4, “Trigger Out Function.” For more information about power management states, see Section 23.4.1, “Register Descriptions.”

Figure 4-5 shows a timing diagram of the POR sequence.

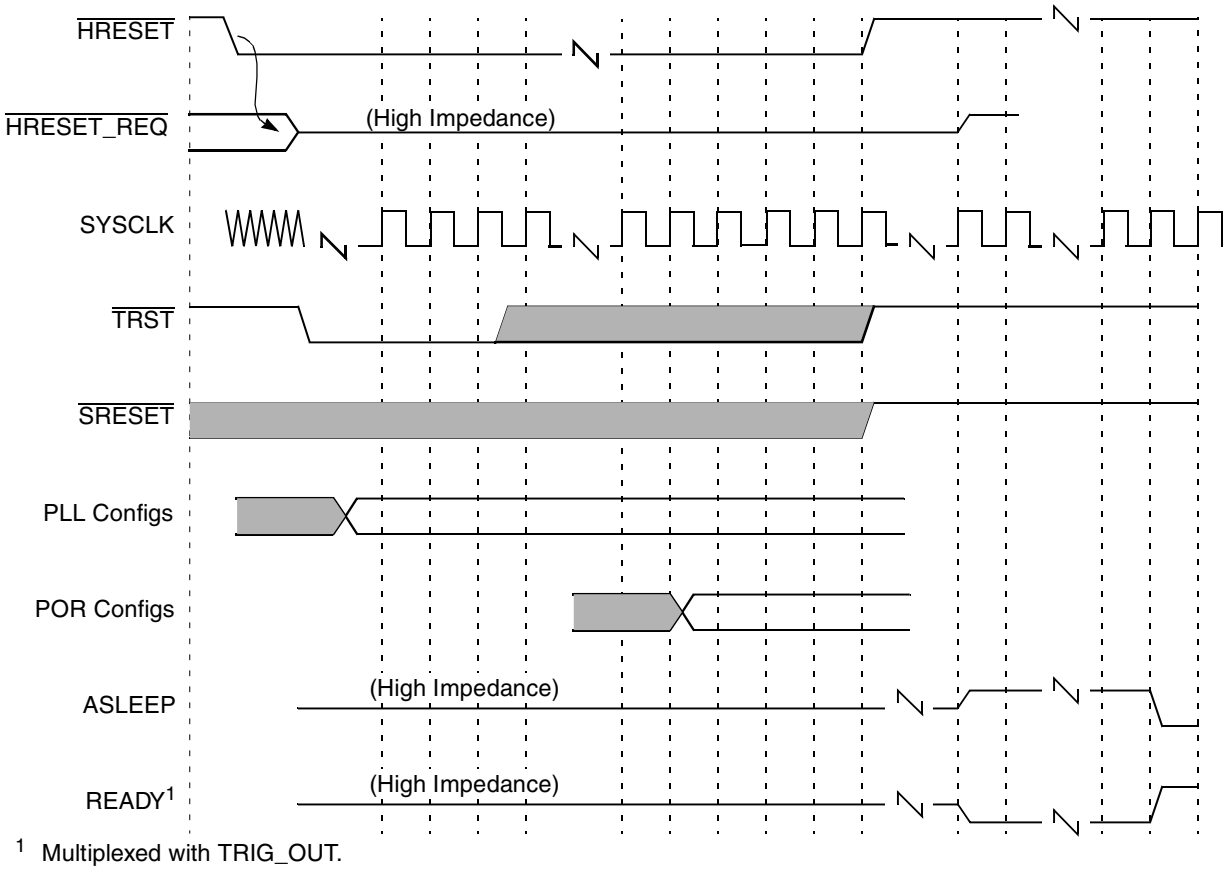


Figure 4-5. Power-On Reset Sequence

4.4.3 Power-On Reset Configuration

Various device functions are initialized by sampling certain signals during the assertion of $\overline{\text{HRESET}}$. The values of all these signals are sampled into registers while $\overline{\text{HRESET}}$ is asserted. These inputs are to be pulled high or low by external resistors. During $\overline{\text{HRESET}}$, all other signal drivers connected to these signals must be in the high-impedance state.

All POR configuration signals have internal pull-up resistors. Refer to the *MPC8610 Integrated Host Processor Hardware Specifications* for proper resistor values to be used for pulling POR configuration signals high or low.

This section describes the functions and modes configured by POR configuration signals. Note that many reset configuration settings are accessible to software through the following read-only memory-mapped registers described in Chapter 23, "Global Utilities:"

- POR PLL status register (PORPLLSR)
- POR boot mode status register (PORBMSR)
- POR device status register (PORDEVSR)
- POR debug mode status register (PORDBGMSR)
- General-purpose POR configuration register (GPPORCR)—reports the value on LAD[0:31] during POR

NOTE

In the following tables, the binary value 0b0 represents a signal pulled down to GND and a value of 0b1 represents a signal pulled up to V_{DD}, regardless of the sense of the functional signal name on the signal.

4.4.3.1 e600 Core PLL Ratio

Table 4-9 describes the e600 core clock PLL ratio configuration inputs that program the core PLL and establish the ratio between the e600 core clock and the platform clock. There is no default value for this PLL ratio; these signals must be pulled to the desired values. Note that the values latched on these signals during POR are not directly accessible in a memory-mapped status register; however, an encoding of the e600 core PLL ratio is accessible through PORPLLSR[e600_RATIO], as described in 23.4.1.1, "POR PLL Status Register (PORPLLSR)," and also in the e600 core HID1 register, as described in Section 6.1.6.2, "Hardware Implementation-Dependent Register 1 (HID1)."

Table 4-9. e600 Core PLL Ratio

Functional Signals	Reset Configuration Name	Value (Binary)	e600 Core: Platform Clock Ratio
LBCTL, LALE, LGPL2/LOE/LFRE, DIU_LD4 No Default	cfg_core_pll[0:3]	1000	2:1
		1010	2.5:1
		1100	3:1
		1110	3.5:1
		All others	Reserved

4.4.3.2 e600 Core Speed

If the platform clock and the e600 core PLL ratio are set to configure the e600 core frequency at a lower speed (less than or equal to 800 MHz), the voltage controlled oscillator (VCO) in the core's PLL must be adjusted to allow the PLL to operate properly. Table 4-10 describes the e600 core speed configuration input. Note that there is no bit associated with cfg_core_speed in any of the global utilities registers.

Table 4-10. e600 Core Speed

Functional Signal	Reset Configuration Name	Value (Binary)	Meaning
ASLEEP Default (1)	cfg_core_speed	0	e600 core frequency at or below 800 MHz
		1	e600 core frequency is above 800 MHz

4.4.3.3 Platform PLL Ratio

The system PLL inputs, shown in Table 4-11, establish the clock ratio between the SYSCLK input and the platform clock used by the device. The platform clock, also called the MPX clock, drives the DDR SDRAM data rate and the e600 core MPX bus interface. There is no default value for this PLL ratio; these signals must be pulled to the desired values. Note that the values latched on these signals during POR are accessible in PORPLLSR[PLAT_RATIO], as described in 23.4.1.1, "POR PLL Status Register (PORPLLSR)."

Table 4-11. Platform Clock PLL Ratio

Functional Signals	Reset Configuration Name	Value (Binary)	Platform Clock : SYSCLK Ratio
DIU_LD[10], LA[28:31] No Default	cfg_sys_pll[0:4]	00011	3 : 1
		00100	4 : 1
		00101	5 : 1
		00110	6 : 1
		00111	7 : 1
		01000	8 : 1
		01001	9 : 1
		01010	10 : 1
		01100	12 : 1
		01110	14 : 1
		01111	15 : 1
		10000	16 : 1
		All others	Reserved

4.4.3.4 eLBC Clock Divisor

The eLBC clock divisor inputs, shown in Table 4-14, set the frequency ratio between the platform clock and the eLBC bus. These inputs are used to set the initial value in the eLBC Clock Ratio Register (LCCR[CLKDIV]) in the eLBC memory mapped configuration registers. Note that the values latched on these signals during POR are accessible in PORPLLSR[eLBC_CLKDIV], as described in 23.4.1.1, "POR PLL Status Register (PORPLLSR)."

Table 4-12. eLBC Clock Divisor

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
LA[25:26] Default (11)	cfg_elbc_clkdiv[0:1]	00	eLBC clock is set to platform clock/4 LCRR[CLKDIV] = 00010
		01	eLBC clock is set to platform clock/8 LCRR[CLKDIV] = 00100
		10	Reserved
		11	eLBC clock is set to platform clock/16 LCRR[CLKDIV] = 01000

For GPCM booting, the maximum division of /16 is recommended.

For NAND-Flash booting through the FCM, the ratio is typically selected so that 66 MHz operation is not exceeded. That is, divided by 4 (00) for platform frequencies up to 333 MHz, and divided by 8 (01) for platform frequencies up to 533 MHz.

4.4.3.5 OCN2 Ratio

The OCN2 ratio input, shown in Table 4-11, establishes the clock ratio between the platform clock and the on-chip network 2 (OCN2) clock. Note that the value latched on this signal during POR is accessible in PORPLLSR[NET2_DIV], as described in 23.4.1.1, “POR PLL Status Register (PORPLLSR).” The clock ratio for platform to OCN1 is always fixed at 2:1.

Table 4-13. OCN2 Ratio

Functional Signal	Reset Configuration Name	Value (Binary)	Meaning
LGPL0/LFCLE Default (1)	cfg_net2_div	0	1:1 ratio
		1	2:1 ratio (required when platform frequency is greater than 400 MHz)

4.4.3.6 CPU Boot Configuration

The CPU boot configuration input, shown in Table 4-14, specifies the boot configuration mode. The value sampled at reset determines if the core is prevented from fetching boot code after reset. If the core is prevented from fetching boot code, it is expected that the device is being configured by an external master. In that case, the final step that the master must take is to free the core to proceed with boot access by setting the MCMPCR[PORT_EN] bit in the MCM port configuration register (MCMPCR) register. See Section 7.4.1.3, “Port Configuration Register (PCR)” for more information.

Note that the value latched on this signal during POR is accessible through PORBMSR[BCFG] described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

Table 4-14. CPU Boot Configuration

Functional Signal	Reset Configuration Name	Value (Binary)	Meaning
LA[27] Default (1)	cfg_cpu_boot	0	CPU boot holdoff mode. The e600 core is prevented from booting until configured by an external master.
		1	The core is allowed to boot without waiting for configuration by an external master.

4.4.3.7 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 I^2C port 1 before the host tries to configure the device. These options also specify normal or extended I^2C addressing modes for both I^2C interfaces. See Section 12.5.7, "Boot Sequencer Mode," for more information on the boot sequencer.

Note that the values latched on these signals during POR are accessible through PORBMSR[BSCFG] described in Section 23.4.1.2, "POR Boot Mode Status Register (PORBMSR)."

Table 4-15. Boot Sequencer Configuration

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
LGPL3/$\overline{\text{LFWP}}$, LGPL5 Default (11)	cfg_boot_seq[0:1]	00	Reserved
		01	Normal I^2C addressing mode is used. Boot sequencer is enabled and loads configuration information from a ROM on the I^2C interface 1. 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 1. A valid ROM must be present.
		11	Boot sequencer is disabled. No I^2C ROM is accessed (default).

NOTE

When the boot sequencer is enabled, the core will be held in reset and thus prevented from fetching boot code until the boot sequencer has completed its task, regardless of the state of the CPU boot configuration signal described in Section 4.4.3.6, "CPU Boot Configuration."

4.4.3.8 Boot ROM Location

This device defines the default boot ROM address range to be 8 Mbytes at address 0x0_FF80_0000 to 0x0_FFFF_FFFF. However, which on-chip peripheral handles these boot ROM accesses can be selected at power up.

The boot ROM location reset configuration inputs, shown in Table 4-16, establish the location of boot ROM. Accesses to the boot vector and the default boot ROM region of the local address map are directed to the interface specified by these inputs.

Note that the values latched on these signals during POR are accessible through PORBMSR[ROM_LOC] described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

Table 4-16. Boot ROM Location

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
DIU_DE, DIU_LD[13:15] Default (1111)	cfg_rom_loc[0:3]	0000–0011	Reserved
		0100	eLBC, FCM, small page, 8-bit NAND Flash ROM
		0101	eLBC, FCM, small page, 16-bit NAND Flash ROM
		0110	eLBC, FCM, large page, 8-bit NAND Flash ROM
		0111	eLBC, FCM, large page, 16-bit NAND Flash ROM
		1000	PCI
		1001	DDR memory controller
		1010	Reserved
		1011	PCI Express 2 (×8)
		1100	PCI Express 1 (×4)
		1101	eLBC, GPCM, 8-bit Flash ROM
		1110	eLBC, GPCM, 16-bit Flash ROM
		1111	eLBC, GPCM, 32-bit Flash ROM (default)

See Section 2.2.6, “Local Address Map Example,” for an example memory map that relies on the default boot ROM values. Also, see Section 4.3.1.3.1, “Boot Page Translation Register (BPTR),” for information on the translation options for the boot page.

4.4.3.9 eLBC ECC Enable

Some systems may require disabling ECC checking on the eLBC interface when booting from the eLBC in FCM mode. Table 4-10 describes the eLBC ECC enable configuration input. The value on this signal is accessible through PORGEN[ELBC_ECC] described in Section 23.4.1.6, “POR General Register (PORGEN).”

Table 4-17. eLBC ECC Enable

Functional Signal	Reset Configuration Name	Value (Binary)	Meaning
DIU_LD0/GPIO[16] Default (1)	cfg_elbc_ecc	0	eLBC ECC checking disabled (BR0[DECC] = 00)
		1	eLBC ECC checking enabled (BR0[DECC] = 01)

4.4.3.10 Alternate Boot Vector

The alternate boot vector input which exists on external signal MDVAL, shown in Table 4-18, allows the processor to boot from PCI Express 1 (×4) outbound translation window 3 by automatically enabling the window and programming the base address to 0x0_FFF0_xxxx and the translated address to

0x0_FFFF_xxxx. This POR configuration option only affects the outbound ATMU window 3 of PCI Express controller 1 (×4). As such, this option only affects booting when PCI Express 1 (×4) is selected as the boot ROM location by the POR option described in Section 4.4.3.8, "Boot ROM Location." Note that the value latched on this signal during POR is accessible through PORBMSR[ABV] described in Section 23.4.1.2, "POR Boot Mode Status Register (PORBMSR)."

Table 4-18. Alternate Boot Vector Location

Functional Signal	Reset Configuration Name	Value (Binary)	Meaning
MDVAL Default (1)	cfg_boot_vec	0	PCI Express 1 outbound ATMU window 3 is enabled, base address is set to 0x0_FFF0_xxxx and translation address is set to 0x0_FFFF_xxxx.
		1	Boot vector fetched from default boot ROM location as described in Section 4.4.3.8, "Boot ROM Location." (default)

4.4.3.11 Host/Agent Configuration

The host/agent reset configuration inputs, shown in Table 4-19, configure the device to act as a host/root complex or as an agent/endpoint on one of the external interfaces. The PCI interface uses the host/agent terminology and PCI Express uses root complex (RC)/endpoint (EP); the terms are roughly equivalent in meaning. In host/RC mode, the device is immediately enabled to master transactions to the external interface. If the device is an agent/EP on the external interface, then the device is disabled from mastering transactions on that interface until the external host/RC enables it to do so. The external host does this by setting the control registers of the controller appropriately.

Note that the values latched on these signals during POR are accessible through PORBMSR[HA] described in Section 23.4.1.2, "POR Boot Mode Status Register (PORBMSR)."

Table 4-19. Host/Agent Configuration

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning		
			PCI	PCI Express 1 (×4)	PCI Express 2 (×8)
$\overline{\text{LWE}}$/$\overline{\text{LFWE}}$/$\overline{\text{LBS}}$[1:3] Default (111)	cfg_host_agt[0:2]	000	Host mode	EP mode	EP mode
		001	Host mode	RC mode	EP mode
		010	Host mode	EP mode	RC mode
		011	Reserved		
		100	Agent mode	RC mode	EP mode
		101	Agent mode	EP mode	RC mode
		110	Agent mode	RC mode	RC mode
		111	Host mode	RC mode	RC mode

NOTE

If the device is an agent/EP on a particular interface, and the CPU is not in boot holdoff mode (as described in Section 4.4.3.6, "CPU Boot Configuration") then the boot ROM should not be located on the interface where the external host exists, because the device is not enabled to master reads onto that interface.

4.4.3.12 SerDes (I/O) Port Selection

The device can be configured with different I/O (SerDes) ports active. Table 4-20 shows the configuration of I/O ports and bit rates (and required reference clocks) that are possible for the PCI Express interfaces.

Note that the values latched on these signals during POR are accessible through PORDEVSR[IO_SEL] described in Section 23.4.1.4, "POR Device Status Register (PORDEVSR)."

Table 4-20. I/O Port Selection

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
DIU_LD[7:9] Default (111)	cfg_io_ports[0:2]	000	Port 2 active PCI Express 1 (×1/×2/×4) disabled on SerDes port 1 PCI Express 2 (×1/×2/×4/×8) active on SerDes port 2
		001	Port 1 active PCI Express 1 (×1/×2/×4) active on SerDes port 1 PCI Express 2 (×1/×2/×4/×8) disabled on SerDes port 2
		010	Reserved
		011	Reserved
		100	Both ports active PCI Express 1 (×1/×2/×4) active on SerDes port 1 PCI Express 2 (×1/×2/×4/×8) active on SerDes port 2
		101	Reserved
		110	Reserved
		111	Both ports disabled (default) PCI Express 1 (×1/×2/×4) disabled on SerDes port 1 PCI Express 2 (×1/×2/×4/×8) disabled on SerDes port 2

4.4.3.13 DDR SDRAM Type

Low-power DDR SDRAM (or mobile DDR DRAM) requires different initialization from other DDR SDRAM types. DDR1 memories require different voltage levels from DDR2. Table 4-21 describes the configuration of the DDR SDRAM type. Note that the value latched on these signals during POR is accessible through PORDEVSR[RTYPE] described in Section 23.4.1.4, "POR Device Status Register (PORDEVSR)."

Table 4-21. DDR SDRAM Type

Functional Signal	Reset Configuration Name	Value (Binary)	Meaning
DIU_LD[11:12] Default (11)	cfg_dram_type[0:1]	00	Reserved
		01	DDR1 2.5V, CKE low at reset
		10	Reserved
		11	DDR2 1.8V, CKE low at reset

4.4.3.14 PCI Clock Select

Note that the values latched on this signal during POR are accessible through PORPLLSR[PCI_CLK_SEL] described in 23.4.1.1, “POR PLL Status Register (PORPLLSR).”

Table 4-22. PCI Clock Selection

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
LGPL1/LFALE Default (1)	cfg_pci_clk	0	Asynchronous—PCI interface clocked by PCI_CLK input.
		1	Synchronous—PCI interface clocked by SYSCLK. **Note:** 2:1 platform clock to SYSCLK ratio is not supported when PCI interface is clocked by SYSCLK.

4.4.3.15 PCI Speed

Note that the values latched on this signal during POR are accessible through PORDEVSR[PCI_SPD] described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

Table 4-23. PCI Speed Selection

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
$\overline{\text{LWE0}}$/$\overline{\text{LFWE0}}$/$\overline{\text{LBS0}}$ Default (1)	cfg_pci_speed	0	PCI frequency is below 33 MHz
		1	PCI frequency is at or above 33 MHz

4.4.3.16 PCI I/O Impedance

Note that the values latched on this signal during POR are accessible through PORIMPSCR[PCI_Z] described in Section 23.4.1.3, “POR I/O Impedance Control Register (PORIMPCR).”

Table 4-24. PCI Impedance Selection

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
DIU_VSYNC Default (1)	cfg_pci_impd	0	Low impedance (25 Ω) drivers are used on the PCI interface.
		1	High impedance (42 Ω) drivers are used on the PCI interface.

4.4.3.17 PCI Arbiter

Note that the values latched on this signal during POR are accessible through PORDEVSR[PCI_ARB] described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

Table 4-25. PCI Arbiter Configuration

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
DIU_HSYNC Default (1)	cfg_pci_arb	0	On-chip PCI arbiter is disabled
		1	On-chip PCI arbiter is enabled

4.4.3.18 Watchdog Timer Enable

Note that the values latched on this signal during POR are accessible through PORDEVSR[PCI_CLK_SEL] described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

Table 4-26. Watchdog Timer Configuration

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
$\overline{\text{UART\_RTS0}}$ Default (1)	cfg_wdt_en	0	Watchdog timer is disabled out of reset
		1	Watchdog timer is enabled out of reset

4.4.3.19 LA Function Configuration

The LA[10:21] signals may be used as latched address signals for the eLBC or as by the SSI block. This configuration signal is used to select the LA[10:21] function at reset. Note that the values latched on this signal during POR are accessible through PORDEVSR[LA_SEL] described in Section 23.4.1.2, “POR Boot Mode Status Register (PORBMSR).”

Table 4-27. LA Function Configuration

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
Default (1)	cfg_ssi_la_sel	0	LA[10:21] are used for eLBC
		1	LA[10:21] are used for SSI

4.4.3.20 General-Purpose POR Configuration

The eLBC address/data bus inputs, shown in Table 4-28, configure the value of the general-purpose POR configuration register defined in Section 23.4.1.4, "POR Device Status Register (PORDEVSR)." This register is intended to facilitate POR configuration of user systems. A value placed on LAD[0:31] during POR is captured and stored (read only) in the GPPORCR. Software can then use this value to inform the operating system about initial system configuration. Typical interpretations include circuit board type, board ID number, or a list of available peripherals.

Table 4-28. General-Purpose POR Configuration

Functional Signals	Reset Configuration Name	Value (Binary)	Meaning
LAD[0:31] No default	cfg_gpinput[0:31]	xx	32 bit general-purpose POR configuration vector to be placed in GPPORCR[POR_CFG_VEC]. On reset, samples the 32-bit LAD bus and stores this sampled value into GPPORCR[POR_CFG_VEC]. Provides a pin sampling capability for the user which, rather than configuring the device, simply stores the state on reset into a 32-bit latch for use by software.

4.4.3.21 Memory Debug Configuration

The memory debug configuration input, shown in Table 4-29, selects which debug outputs (DDR or LBC memory controller) are driven onto the MSRCID[0:4] and MDVAL debug signals. Note that the value latched on this signal during POR is accessible through PORDBGMSR[MEM_SEL] described in Section 23.4.1.4, "POR Device Status Register (PORDEVSR)."

Table 4-29. Memory Debug Configuration

Functional Signal	Reset Configuration Name	Value (Binary)	Meaning
MSRCID0	cfg_mem_debug	0	Debug information from the enhanced local bus controller (eLBC) is driven on the MSRCID[0:4] and MDVAL signals
		1	Debug information from the DDR memory controller is driven on the MSRCID[0:4] and MDVAL signals (default).

4.4.4 Clocking

The following paragraphs describe the clocking within the device.

4.4.4.1 System Clock

The device takes a single input clock, SYSCLK, as its primary clock source for the e600 core and all of the devices and interfaces that operate synchronously with the cores. As shown in Figure 4-6, the SYSCLK input (frequency) is multiplied up using a phase-locked loop (PLL) to create the platform clock (also called the MPX clock). The platform clock is used by virtually all of the synchronous system logic and other internal blocks such as the DMA and interrupt controller. The platform clock also feeds the PLL in the e600 core. Note that the divide-by-two clock divider and the divide-by-*n* clock divider, shown in Figure 4-6, are located in the DDR and local bus blocks, respectively.

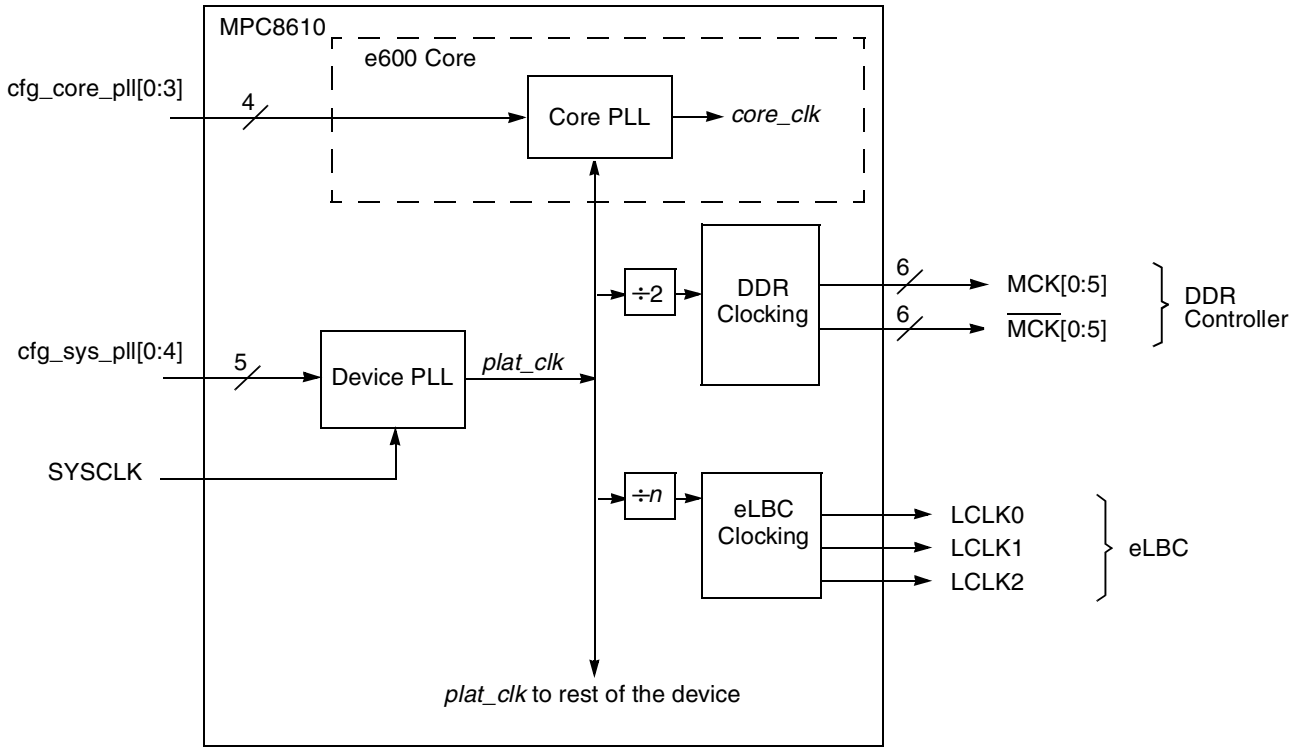


Figure 4-6. Clock Subsystem Block Diagram

4.4.4.2 PCI Express Clocks

Clocks for the high speed interfaces on the MPC8610 are derived from a PLL in each SerDes block. This PLL is driven by a differential reference clock pair (SD*n*_REF_CLK/$\overline{\text{SD}n\text{\_REF\_CLK}}$) whose input frequency is a function of the protocol and bit rate being used as shown in Table 4-30.

Table 4-30. High Speed Interface Clocking

Interfaces	Bit Rate	Reference Clock Frequency
PCI Express	2.5 Gbps	100 MHz

4.4.4.2.1 Minimum Frequency Requirements

Section 4.4.3.12, "SerDes (I/O) Port Selection," describes various high-speed interface configuration options. Note that the MPX clock frequency must be considered for proper operation of such interfaces.

Please refer to the *MPC8610 Integrated Processor Hardware Specifications*, for specific supported frequencies.

Part II
e600 Core

This part describes the many features of the MPC8610 core processor at an overview level. The following chapters are included:

- Chapter 5, “e600 Core Overview,” provides an overview of the e600 core processor and the L1 caches and MMU that, together with the core, comprise the core complex.
- Chapter 6, “e600 Core Registers and Instruction Set Summary,” provides a listing of the e600 registers in reference form.

Chapter 5
e600 Core Overview

This chapter provides an overview of the e600 core features, including a block diagram showing the major functional components.

This document also provides information about how the e600 implementation complies with the PowerPC™ instruction set architecture (ISA) definitions including AltiVec™ extensions.

5.1 e600 Core Overview

This section describes the features and general operation of the e600 core and provides a block diagram showing the major functional units. The e600 implements the PowerPC ISA and is a reduced instruction set computer (RISC) core. The e600 core includes separate 32-Kbyte L1 instruction and data caches and a 256-Kbyte L2 cache. The core is a high-performance superscalar design supporting multiple execution units, including four independent units that execute AltiVec instructions.

The e600 core implements the 32-bit portion of the PowerPC ISA, which provides 32-bit effective addresses, integer data types of 8, 16, and 32 bits, and floating-point data types of 32 (single precision) and 64 (double precision) bits. The core supports up to 4 Petabytes (2^{52}) of virtual memory and up to 64 Gigabytes (2^{36}) of physical memory.

The e600 core also implements the AltiVec instruction set architectural extension. The e600 core can dispatch and complete three instructions simultaneously. It incorporates the following execution units:

- 64-bit floating-point unit (FPU)
- Branch processing unit (BPU)
- Load/store unit (LSU)
- Four integer units (IUs):
  - Three shorter latency IUs (IU1a–IU1c)—execute all integer instructions except multiply, divide, and move to/from special-purpose register (SPR) instructions.
  - Longer latency IU (IU2)—executes miscellaneous instructions including condition register (CR) logical operations, integer multiplication and division instructions, and move to/from SPR instructions.
- Four vector units that support AltiVec instructions:
  - Vector permute unit (VPU)
  - Vector integer unit 1 (VIU1)—performs shorter latency integer calculations
  - Vector integer unit 2 (VIU2)—performs longer latency integer calculations
  - Vector floating-point unit (VFPU)

The ability to execute several instructions in parallel and the use of simple instructions with rapid execution times yield high efficiency and throughput for e600-based systems. Most integer instructions (including VIU1 instructions) have a one-clock cycle execution latency.

Several execution units feature multiple-stage pipelines; that is, the tasks they perform are broken into subtasks executed in successive stages. Typically, instructions follow one another through the stages, so a four-stage unit can work on four instructions when its pipeline is full. So, although an instruction may have to pass through several stages, the execution unit can achieve a throughput of one instruction per clock cycle.

AltiVec computational instructions are executed in four independent, pipelined AltiVec execution units. A maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). This means an instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions. Because VPU has a two-stage pipeline; the VIU2 and VFPU each have four-stage pipelines, as many as ten AltiVec instructions can be executing concurrently.

Note that for the e600 core, double- and single-precision versions of floating-point instructions have the same latency. For example, a floating-point multiply-add instruction takes 5 cycles to execute, regardless of whether it is single (**fmadds**) or double precision (**fmadd**).

The e600 core has independent 32-Kbyte, eight-way set-associative, physically addressed L1 (level one) caches for instructions and data, and independent instruction and data memory management units (MMUs). Each MMU has a 128-entry, two-way set-associative translation lookaside buffer (DTLB and ITLB) that saves recently used page address translations. Block address translation is implemented with the eight-entry instruction and data block address translation (IBAT and DBAT) arrays defined by the PowerPC ISA. During block translation, effective addresses are compared simultaneously with all BAT entries, as described in the “Memory Management” chapter of the *e600 PowerPC Core Reference Manual*. For information about the L1 caches, see the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*.

The L2 cache is implemented with 256-Kbyte, eight-way set-associative physically addressed memory within the core available for storing data, instructions, or both. The L2 cache supports parity generation and checking for both tags and data. If error checking and correction (ECC) is disabled, it responds with an 11.5-cycle load latency for an L1 miss that hits in the L2; if ECC is enabled, the L2 load access time is 12.5 cycles. The L2 cache is fully pipelined for two-cycle throughput. For information about the L2 cache implementation, see the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*.

The two power-saving modes, nap and sleep, progressively reduce power dissipation. When functional units are idle, a dynamic power management mode causes those units to enter a low-power mode automatically without affecting operational performance, software execution, or hardware external to the core. Section 5.2.7, “Power and Thermal Management,” describes how power management can be used to reduce power consumption when the core, or portions of it, are idle. It also describes how the instruction cache throttling mechanism reduces the instruction dispatch rate. Power management states are described more fully in the “Power and Thermal Management” chapter of the *e600 PowerPC Core Reference Manual*.

The performance monitor facility 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 5.2.8, “Core Performance Monitor,” describes the operation of the performance monitor diagnostic tool. This functionality is fully described in the “Performance Monitor” chapter of the *e600 PowerPC Core Reference Manual*.

Figure 5-1 shows the parallel organization of the execution units (shaded in the diagram). Note that this is a conceptual model showing basic features, rather than an attempt at showing how features are implemented physically.

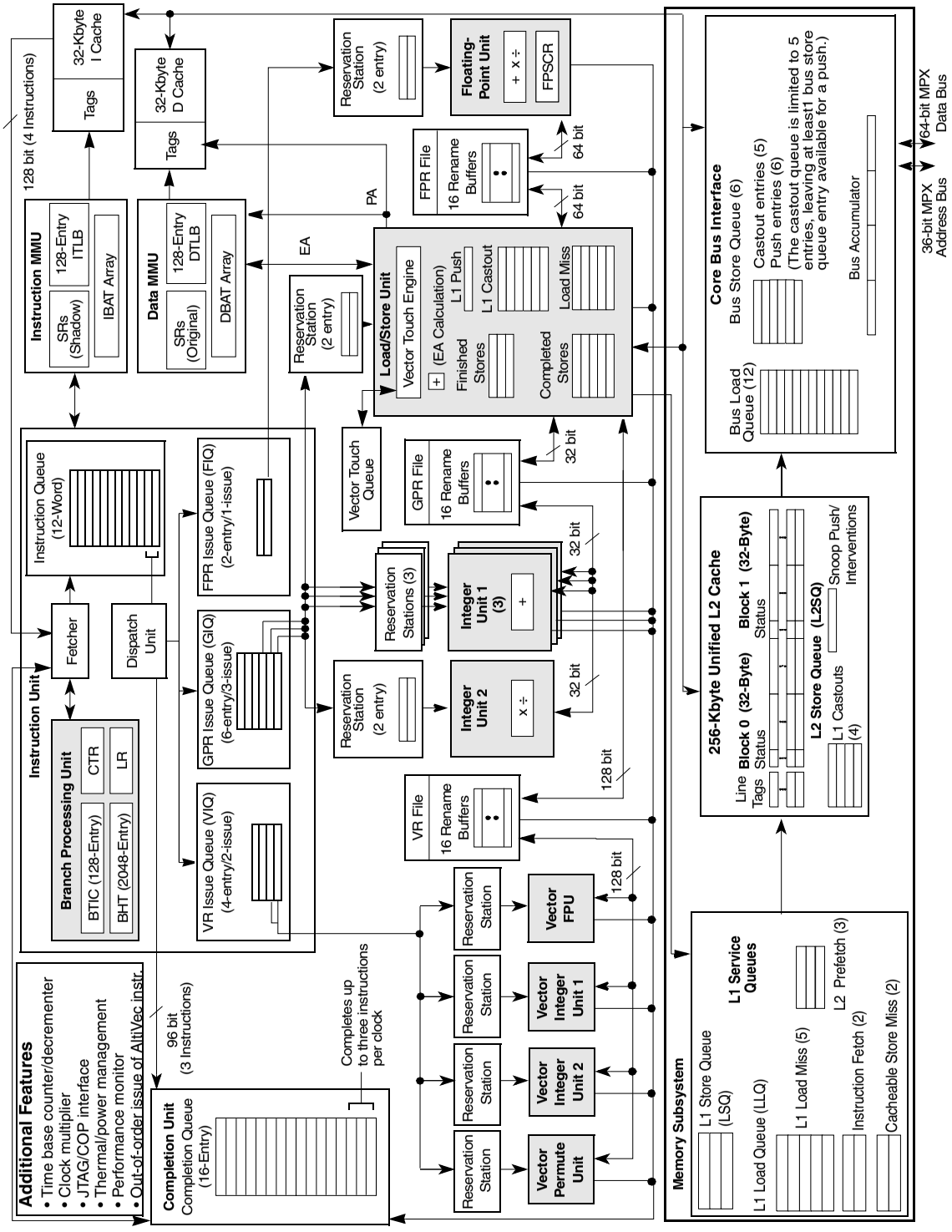


Figure 5-1. e600 Core Block Diagram

5.2 e600 Core Features

This section describes the features of the e600 core. The interrelationships of these features are shown in Figure 5-1.

Major features of the e600 core are as follows:

- High-performance superscalar e600 core
  - As many as 4 instructions can be fetched from the instruction cache at a time.
  - As many as 3 instructions can be dispatched to the issue queues at a time.
  - As many as 12 instructions can be in the instruction queue (IQ).
  - As many as 16 instructions can be at some stage of execution simultaneously.
  - Single-cycle execution for most instructions
  - One-instruction throughput per clock cycle for most instructions
  - Seven-stage pipeline control
- Eleven independent execution units and three register files
  - Branch processing unit (BPU) features static and dynamic branch prediction
    - 128-entry (32-set, four-way set-associative) branch target instruction cache (BTIC), a cache of branch instructions that have been encountered in branch/loop code sequences. If a target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, a fetch that hits the BTIC provides the first 4 instructions in the target stream.
    - 2048-entry branch history table (BHT) with 2 bits per entry for four levels of prediction—strongly not-taken, not-taken, taken, strongly taken
    - Up to three outstanding speculative branches
    - Branch instructions that do not update the count register (CTR) or link register (LR) are often removed from the instruction stream.
    - Eight-entry link register stack to predict the target address of Branch Conditional to Link Register (**bclr**) instructions
  - Four integer units (IUs) that share 32 GPRs for integer operands
    - Three identical IUs (IU1a, IU1b, and IU1c) can execute all integer instructions except multiply, divide, and move to/from SPR instructions.
    - IU2 executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions.
  - 64-bit floating-point unit (FPU)
    - Five-stage FPU
    - Designed to comply with IEEE Std. 754™-1985 FPU for both single- and double-precision operations
    - Supports non-IEEE Std. 754 mode for time-critical operations
    - Hardware support for denormalized numbers
    - Thirty-two 64-bit FPRs for single- or double-precision operands

— Four vector units and 32-entry vector register file (VRs)
  – Vector permute unit (VPU)
  – Vector integer unit 1 (VIU1) handles short-latency AltiVec integer instructions, such as vector add instructions (for example, **vaddsbs**, **vaddshs**, and **vaddsws**)
  – Vector integer unit 2 (VIU2) handles longer-latency AltiVec integer instructions, such as vector multiply add instructions (for example, **vmhaddshs**, **vmhraddshs**, and **vmladduhm**).
  – Vector floating-point unit (VFPU)
— Three-stage load/store unit (LSU)
  – Supports integer, floating-point and vector instruction load/store traffic
  – Four-entry vector touch queue (VTQ) supports all four architected AltiVec data stream operations
  – Three-cycle GPR and AltiVec load latency (byte, half word, word, vector) with single-cycle throughput
  – Four-cycle FPR load latency (single, double) with single-cycle throughput
  – No additional delay for misaligned access within double-word boundary
  – Dedicated adder calculates effective addresses (EAs)
  – Supports store gathering
  – Performs alignment, normalization, and precision conversion for floating-point data
  – Executes cache control and TLB instructions
  – Performs alignment, zero padding, and sign extension for integer data
  – Supports hits under misses (multiple outstanding misses)
  – Supports both big-endian and PPC_LE modes, including misaligned little-endian accesses

- Dispatch unit—The decode/dispatch stage fully decodes each instruction.
- The FIQ (floating-point issue queue), VIQ (vector issue queue), and GIQ (general purpose issue queue) can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  — Instructions can be dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  — A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  — Space must be available in the completion queue (CQ) for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not in an issue queue).
- Rename buffers
  — 16 GPR (general purpose register) rename buffers
  — 16 FPR (floating-point register) rename buffers
  — 16 VR (vector register) rename buffers
- Completion unit
  — Retires an instruction from the 16-entry CQ when all instructions ahead of it have been completed, the instruction has finished execution, and no interrupts are pending
  — Guarantees sequential programming model (precise interrupt model)

- Monitors all dispatched instructions and retires them in order
- Tracks unresolved branches and flushes instructions after a mispredicted branch
- Retires as many as three instructions per clock cycle

- L1 cache has the following characteristics:
  - Two separate 32-Kbyte instruction and data caches (Harvard architecture)
  - Instruction and data caches are eight-way set-associative
  - Instruction and data caches have 32-byte cache blocks. A cache block is the block of memory that a coherency state describes—it corresponds to a cache line for the L1 data cache.
  - Cache directories are physically addressed. The physical (real) address tag is stored in the cache directory.
  - The caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way.
  - Cache write-back or write-through operation is programmable on a per-page or per-block basis.
  - Instruction cache can provide four instructions per clock cycle; data cache can provide four words per clock cycle
    - Two-cycle latency and single-cycle throughput for instruction or data cache accesses
  - Caches can be disabled in software
  - Caches can be locked in software
  - Supports a four-state modified/exclusive/shared/invalid (MESI) coherency protocol
    - A single coherency status bit for each instruction cache block allows encoding for the following two possible states:

      Invalid (INV)

      Valid (VAL)
    - Three status bits for each data cache block allow encoding for coherency, as follows:

      0xx = invalid (I)

      101 = shared (S)

      100 = exclusive (E)

      110 = modified (M)
  - Separate copy of data cache tags for efficient snooping
  - Both L1 caches support parity generation and checking (enabled through bits in the instruction cache and interrupt control (ICTRL) register) as follows:
    - Instruction cache—one parity bit per instruction
    - Data cache—one parity bit per byte of data
  - No snooping of instruction cache except for **icbi** instruction
  - Caches implement a pseudo least-recently-used (PLRU) replacement algorithm within each way
  - Data cache supports AltiVec LRU and transient instructions, as described in Section 5.3.2.2, “AltiVec Instruction Set”

  - Critical double- and/or quad-word forwarding is performed as needed. Critical quad-word forwarding is used for AltiVec loads and instruction fetches. Other accesses use critical double-word forwarding.
- Level 2 (L2) cache within the core has the following features:
  - Integrated 256-Kbyte, eight-way set-associative unified instruction and data cache
  - Pipelined to provide 32 bytes every other clock cycle to the L1 caches
  - Total latency of 11.5 processor cycles for L1 data cache miss that hits in the L2 with ECC disabled, 12.5 cycles when ECC is enabled
  - Uses one of two random replacement algorithms (selectable through L2CR)
  - Cache write-back or write-through operation programmable on a per-page or per-block basis
  - Organized as 32 bytes/block and 2 blocks (sectors)/line (a cache block is the block of memory that a coherency state describes).
  - Supports error correction and detection using a SECDED (single-error correction, double-error detection) protocol. Every 64 bits of data comes with 8 bits of error detection/correction, which can be programmed as ECC across the 64 bits of data, byte parity, or no error detection/correction.
  - Supports parity generation and checking for both tags and data (enabled through L2CR). Tag parity is enabled separately in the L2ERRDIS register, and data parity can be enabled through L2CR only when ECC is disabled.
  - Error injection modes provided for testing
- Separate memory management units (MMUs) for instructions and data
  - 52-bit virtual address; 32- or 36-bit physical address
  - Address translation for 4-Kbyte pages, variable-sized blocks, and 256-Mbyte segments
  - Memory programmable as write-back/write-through, caching-inhibited/caching-allowed, and memory coherency enforced/memory coherency not enforced on a page or block basis
  - Separate IBATs and DBATs (eight each) also defined as SPRs
  - Separate instruction and data translation lookaside buffers (TLBs)
    - Both TLBs are 128-entry, two-way set-associative, and use LRU replacement algorithm
    - TLBs are hardware or software reloadable (that is, on a TLB miss a page table search is performed in hardware or by system software).
- Efficient data flow
  - Although the VR/LSU interface is 128 bits, the L1/L2 bus interface allows up to 256 bits.
  - The L1 data cache is pipelined to provide 128 bits/cycle to or from the VRs.
  - L2 cache is fully pipelined to provide 32 bytes every other processor clock cycle to the L1 cache
  - As many as nine outstanding, out-of-order cache misses are allowed between the L1 data cache and L2 bus: up to 5 load or touch instructions, 2 cacheable store instructions, and/or 2 instruction fetches.
  - As many as 8 out-of-order transactions can be present on the MPX bus.

  - Store merging for multiple store misses to the same line. Only coherency action taken (address-only) for store misses merged to all 32 bytes of a cache block (no data tenure needed)
  - Support for a second cacheable store miss
  - Three-entry finished store queue and five-entry completed store queue between the LSU and the L1 data cache
  - Separate additional queues for efficient buffering of outbound data (such as castouts and write-through stores) from the L1 data cache and L2 cache
- Multiprocessing support features include the following:
  - Hardware-enforced, MESI cache coherency protocols for data cache
  - Load/store with reservation instruction pair for atomic memory references, semaphores, and other multiprocessor operations
- Power and thermal management
  - The following two power-saving modes are available to the system:
    - Nap—Instruction fetching is halted. Only those clocks for the time base, decrementer, and JTAG logic remain running. The core goes into the doze state to snoop memory operations on the MPX bus and then back to nap using a $\overline{qreq}$/$\overline{qack}$ processor-system handshake protocol.
    - Sleep—Power consumption is further reduced by disabling bus snooping, leaving only the PLL in a locked and running state. All internal functional units are disabled.
  - Instruction cache throttling provides control of instruction fetching to lower device temperature.
- Performance monitor helps to debug system designs and improve software efficiency
- In-system testability and debugging features through JTAG boundary-scan capability
- Reliability and serviceability
  - Parity checking on L1 and L2 cache arrays

5.2.1 Instruction Flow

As shown in Figure 5-1, the e600 core instruction unit provides centralized control of instruction flow to the execution units. The instruction unit contains a sequential fetcher, 12-entry instruction queue (IQ), dispatch unit, and branch processing unit (BPU). It determines the address of the next instruction to be fetched based on information from the sequential fetcher and from the BPU.

See the “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual* for a detailed discussion of instruction timing.

The sequential fetcher loads instructions from the instruction cache into the instruction queue. The BPU extracts branch instructions from the sequential fetcher. Branch instructions that cannot be resolved immediately are predicted using either e600-specific dynamic branch prediction or architecture-defined static branch prediction.

Branch instructions that do not affect the LR or CTR are often removed from the instruction stream. The “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual* describes when a branch can be removed from the instruction stream.

Instructions dispatched beyond a predicted branch do not complete execution until the branch is resolved, preserving the programming model of sequential execution. If branch prediction is incorrect, the instruction unit flushes all predicted path instructions, and instructions are fetched from the correct path.

5.2.1.1 Instruction Queue and Dispatch Unit

The instruction queue (IQ), shown in Figure 5-1 holds as many as 12 instructions and loads as many as 4 instructions from the instruction cache during a single processor clock cycle.

The fetcher attempts to initiate a new fetch every cycle. The two fetch stages are pipelined, so as many as four instructions can arrive to the IQ every cycle. All instructions except branch (**b***x*), Return from Interrupt (**rfi**), System Call (**sc**), Instruction Synchronize (**isync**), and no-op instructions are dispatched to their respective issue queues from the dispatch entries in the instruction queue (IQ0–IQ2) at a maximum rate of three instructions per clock cycle. Reservation stations are provided for the three IU1s, IU2, FPU, LSU, VPU, VIU2, VIU1, and VFPU. The dispatch unit checks for source and destination register dependencies, determines whether a position is available in the CQ, and inhibits subsequent instruction dispatching as required.

Branch instruction can be detected, decoded, and predicted from entries IQ0–IQ7. See the “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual.*

5.2.1.2 Branch Processing Unit (BPU)

The BPU receives branch instructions from the IQ and executes them early in the pipeline, achieving the effect of a zero-cycle branch in some cases.

Branches with no outstanding dependencies (CR, LR, or CTR unresolved) can be processed and resolved immediately. For branches in which only the direction is unresolved due to a CR or CTR dependency, the branch path is predicted using either architecture-defined static branch prediction or e600-specific dynamic branch prediction. Dynamic branch prediction is enabled if HID0[BHT] is set. For **bclr** branches

where the target address is unresolved due to an LR dependency, the branch target can be predicted using the hardware link stack. Link stack prediction is enabled if HID0[LRSTK] is set.

When a prediction is made, instruction fetching, dispatching, and execution continue from the predicted path, but instructions cannot complete and write back results to architected registers until the prediction is determined to be correct (resolved). When a prediction is incorrect, the instructions from the incorrect path are flushed from the core and processing begins from the correct path.

Dynamic prediction is implemented using a 2048-entry branch history table (BHT), a cache that provides two bits per entry that together indicate four levels of prediction for a branch instruction—not-taken, strongly not-taken, taken, strongly taken. When dynamic branch prediction is disabled, 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 e600 core executes instructions from the predicted target stream although the results are not committed to architected registers until the conditional branch is resolved. Unresolved branches are held in a three-entry branch queue. When the branch queue is full, no further conditional branches can be processed until one of the conditions in the branch queue is resolved.

When a branch is taken or predicted as taken, instructions from the untaken path must be flushed and the target instruction stream must be fetched into the IQ. The BTIC is a 128-entry, four-way set associative cache that contains the most recently used branch target instructions (up to four instructions per entry) for **b** and **bc** branches. When a taken branch instruction of this type hits in the BTIC, the instructions arrive in the IQ two clock cycles later, a clock cycle sooner than they would arrive from the instruction cache. Additional instructions arrive from the instruction cache in the next clock cycle. The BTIC reduces the number of missed opportunities to dispatch instructions and gives the core a 1-cycle head start on processing the target stream.

The BPU contains an adder to compute branch target addresses and three user-accessible registers—the link register (LR), the count register (CTR), and the condition register (CR). The BPU calculates the return pointer for subroutine calls and saves it in the LR for certain types of branch instructions. The LR also contains the branch target address for Branch Conditional to Link Register (**bclr***x*) instructions. The CTR contains the branch target address for Branch Conditional to Count Register (**bcctr***x*) instructions. Because the LR and CTR are SPRs, their contents can be copied to or from any GPR. Also, 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.

5.2.1.3 Completion Unit

The completion unit operates closely with the instruction unit. Instructions are fetched and dispatched in program order. At the point of dispatch, the program order is maintained by assigning each dispatched instruction a successive entry in the 16-entry CQ. The completion unit tracks instructions from dispatch through execution and retires them in program order from the three bottom CQ entries (CQ0–CQ2).

Instructions cannot be dispatched to an execution unit unless there is a CQ vacancy.

Branch instructions that do not update the CTR or LR are often removed from the instruction stream. Those that are removed do not take a CQ entry. Branches that are not removed from the instruction stream

follow the same dispatch and completion procedures as non-branch instructions but are not dispatched to an issue queue.

Completing an instruction commits execution results to architected registers (GPRs, FPRs, VRs, LR, and CTR). In-order completion ensures the correct architectural state when the e600 core must recover from a mispredicted branch or any interrupt. An instruction is retired as it is removed from the CQ.

For a more detailed discussion of instruction completion, see the “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual*.

5.2.1.4 Independent Execution Units

In addition to the BPU, the e600 core provides the ten execution units described in the following sections.

5.2.1.4.1 AltiVec Vector Permute Unit (VPU)

The VPU executes permutation instructions such as pack, unpack, merge, splat, and permute on vector operands.

5.2.1.4.2 AltiVec Vector Integer Unit 1 (VIU1)

The VIU1 executes simple vector integer computational instructions, such as addition, subtraction, maximum and minimum comparisons, averaging, rotation, shifting, comparisons, and Boolean operations.

5.2.1.4.3 AltiVec Vector Integer Unit 2 (VIU2)

The VIU2 executes longer-latency vector integer instructions, such as multiplication, multiplication/addition, and sum-across with saturation.

5.2.1.4.4 AltiVec Vector Floating-Point Unit (VFPU)

The VFPU executes all vector floating-point instructions.

A maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

5.2.1.4.5 Integer Units (IUs)

The integer units (three IU1s and IU2) are shown in Figure 5-1. The IU1s execute shorter latency integer instructions, that is, all integer instructions except multiply, divide, and move to/from special-purpose register instructions. IU2 executes integer instructions with latencies of three cycles or more.

IU2 has a 32-bit integer multiplier/divider and a unit for executing CR logical operations and move to/from SPR instructions. The multiplier supports early exit for operations that do not require full 32×32-bit multiplication.

5.2.1.4.6 Floating-Point Unit (FPU)

The FPU, shown in Figure 5-1, is designed such that double-precision operations require only a single pass, with a latency of 5 cycles. As instructions are dispatched to the FPUs reservation station, source operand data can be accessed from the FPRs or from the FPR rename buffers. Results in turn are written to the rename buffers and made available to subsequent instructions. Instructions start execution from the bottom reservation station only and execute in program order.

The FPU contains a single-precision multiply-add array and the floating-point status and control register (FPSCR). The multiply-add array allows the e600 core to implement multiply and multiply-add operations efficiently. The FPU is pipelined so that one single- or double-precision instruction can be issued per clock cycle.

Note that an execution bubble occurs after four consecutive, independent floating-point arithmetic instructions execute to allow for a normalization special case. Thirty-two 64-bit floating-point registers are provided to support floating-point operations. Stalls due to contention for FPRs are minimized by automatic allocation of the 16 floating-point rename registers. The e600 core writes the contents of the rename registers to the appropriate FPR when floating-point instructions are retired by the completion unit.

The e600 core supports all IEEE Std. 754 floating-point data types (normalized, denormalized, NaN, zero, and infinity) in hardware, eliminating the latency incurred by software interrupt routines.

5.2.1.4.7 Load/Store Unit (LSU)

The LSU executes all load and store instructions as well as the AltiVec LRU and transient instructions and provides the data transfer interface between the GPRs, FPRs, VRs, and the cache/core memory subsystem (MSS). The LSU also calculates effective addresses and aligns data.

Load and store instructions are issued and translated in program order; however, some memory accesses can occur out of order. Synchronizing instructions can be used to enforce strict ordering. When there are no data dependencies and the guarded bit for the page or block is cleared, a maximum of one out-of-order cacheable load operation can execute per clock cycle from the perspective of the LSU. Loads to FPRs require a 4-cycle total latency. Data returned from the cache is held in a rename register until the completion logic commits the value to a GPR, FPR, or VR. Stores cannot be executed out of order and are held in the store queue until the completion logic signals that the store operation is to be completed to memory. The e600 core executes store instructions with a maximum throughput of one every three clock cycles and a 3-cycle total latency to the data cache. The time required to perform the load or store operation depends on the processor: bus clock ratio and whether the operation involves the caches, system memory, or an I/O device.

5.2.2 Memory Management Units (MMUs)

The e600 core MMUs support up to 4 Petabytes (2^{52}) of virtual memory and 64 Gigabytes (2^{36}) of physical memory for instructions and data. The MMUs control access privileges for these spaces on block and page granularities. Referenced and changed status is maintained by the core for each page to support demand-paged virtual memory systems. The MMUs are contained within the LSU.

The MMU translates the effective address calculated by the IU and LSU to determine the physical address for the memory access.

The e600 core supports the following types of memory translation:

- Real addressing mode—In this mode, translation is disabled by clearing bits in the machine state register (MSR): MSR[IR] for instruction fetching or MSR[DR] for data accesses. When address translation is disabled, the physical address is identical to the effective address. When extended addressing is disabled (HID0[XAEN] = 0) a 32-bit physical address is used, PA[4–35]. For more details, see the “Memory Management” chapter of the *e600 PowerPC Core Reference Manual.*
- Page address translation—translates the page frame address for a 4-Kbyte page size
- Block address translation—translates the base address for blocks (4 Gbytes)

If translation is enabled, the appropriate MMU translates the higher-order bits of the effective address into physical address bits. Lower-order address bits are untranslated and are the same for both logical and physical addresses. These bits are directed to the caches where they form the index into the eight-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- or 36-bit physical address is used by the core memory subsystem (MSS) and the bus interface unit (BIU) to access memory external to the core.

The TLBs store page address translations for recent memory accesses. For each access, an effective address is presented for page and block translation simultaneously. If a translation is found in both the TLB and the BAT array, the block address translation in the BAT array is used. Usually the translation is in a TLB and the physical address is readily available to the cache. When a page address translation is not in a TLB, hardware or system software searches for one in the page table following the model defined by the PowerPC ISA.

Instruction and data TLBs provide address translation in parallel with the cache access, incurring no additional time penalty in the event of a TLB hit. The e600 core instruction and data TLBs are 128-entry, two-way set-associative caches that contain address translations. The core can initiate a hardware or system software search of the page tables in memory on a TLB miss.

5.2.3 L1 Instruction and Data Caches Within the Core

The e600 core implements separate L1 instruction and data caches. Each cache is 32 Kbytes and eight-way set-associative. As defined by the PowerPC ISA, they are physically indexed. Each cache block contains eight contiguous words from memory that are loaded from an eight-word boundary (that is, bits EA[27–31] are zeros); thus, a cache block never crosses a page boundary. An entire cache block can be updated by a four-beat burst load across a 64-bit MPX bus. Misaligned accesses across a page boundary can incur a performance penalty. The data cache is a non-blocking, write-back cache with hardware support for reloading on cache misses. The critical double word is transferred on the first beat and is forwarded to the requesting unit, minimizing stalls due to load delays. For vector loads, the critical quad word is handled similarly but is transferred on the second beat. The cache being loaded is not blocked to internal accesses while the load completes.

The e600 core L1 cache organization is shown in Figure 5-2.

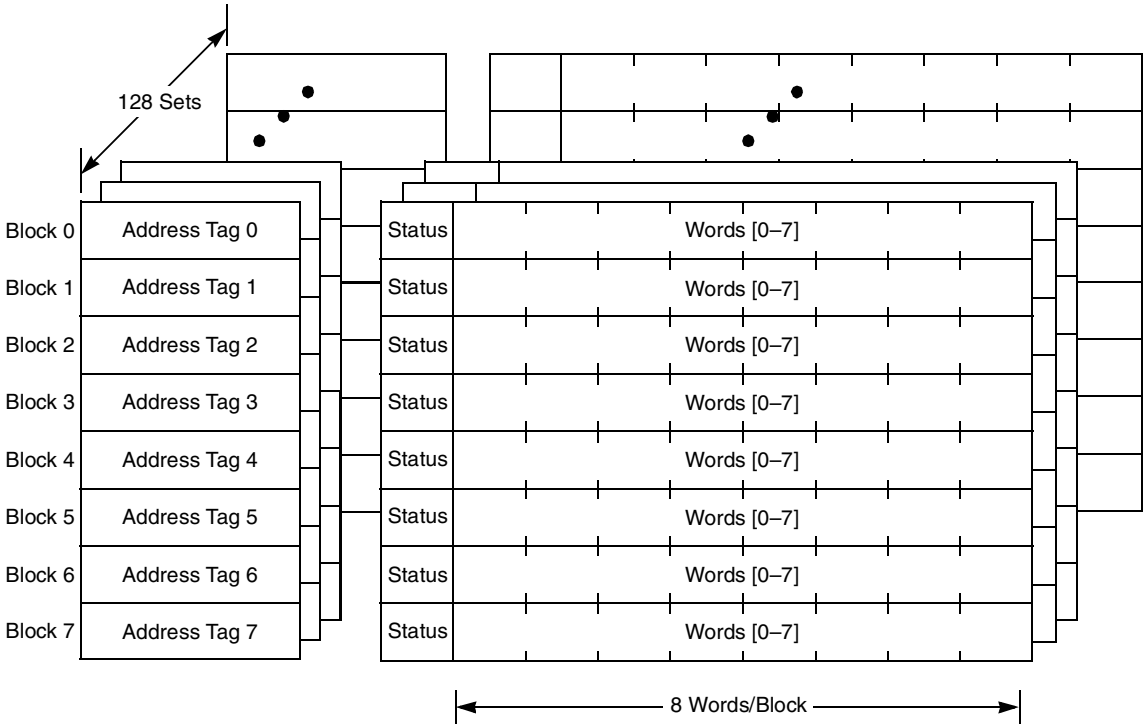


Figure 5-2. L1 Cache Organization

The instruction cache provides up to four instructions per clock cycle to the instruction queue. The instruction cache can be invalidated entirely or on a cache-block basis. It is invalidated and disabled by setting HID0[ICFI] and then clearing HID0[ICE]. The instruction cache can be locked by setting HID0[ILOCK]. The instruction cache supports only the valid/invalid states.

The data cache provides four words per clock cycle to the LSU. Like the instruction cache, the data cache can be invalidated all at once or on a per-cache-block basis. The data cache can be invalidated and disabled by setting HID0[DCFI] and then clearing HID0[DCE]. The data cache can be locked by setting HID0[DLOCK]. The data cache tags are dual-ported, so a load or store can occur simultaneously with a snoop.

The e600 core also implements a 128-entry (32-set, four-way set-associative) branch target instruction cache (BTIC). The BTIC is a cache of branch instructions that have been encountered in branch/loop code sequences. If the target instruction is in the BTIC, it is fetched into the instruction queue a cycle sooner than it can be made available from the instruction cache. Typically, the BTIC contains the first four instructions in the target stream.

The BTIC can be disabled and invalidated through software. As with other aspects of e600 core instruction timing, BTIC operation is optimized for cache-line alignment. If the first target instruction is one of the first five instructions in the cache block, the BTIC entry holds four instructions. If the first target instruction is the last instruction before the cache block boundary, it is the only instruction in the corresponding BTIC entry. If the next-to-last instruction in a cache block is the target, the BTIC entry holds two valid target instructions, as shown in Figure 5-3.

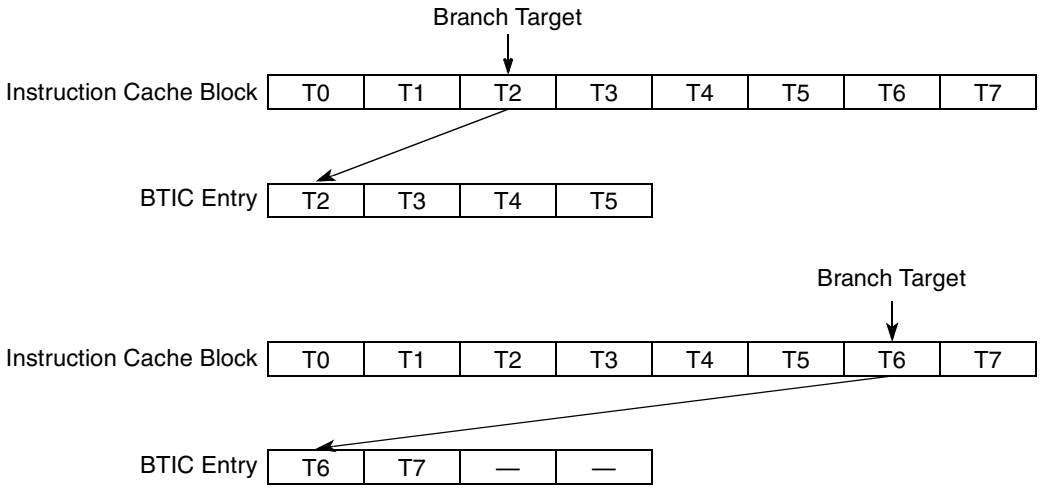


Figure 5-3. Alignment of Target Instructions in the BTIC

BTIC ways are updated using a FIFO algorithm.

For more information and timing examples showing cache hit and cache miss latencies, see the "Instruction Timing" chapter of the *e600 PowerPC Core Reference Manual.*

5.2.4 L2 Cache Implementation

The integrated L2 cache is a unified 256-Kbyte cache that receives memory requests from both the L1 instruction and data caches independently. It is eight-way set-associative and organized with 32-byte blocks and two blocks/line.

Each line consists of 64 bytes of data organized as two blocks (also called sectors). Although all 16 words in a cache line share the same address tag, each block maintains the three separate status bits for the 8 words of the cache block, the unit of memory at which coherency is maintained. Thus, each cache line can contain 16 contiguous words from memory that are read or written as 8-word operations.

The integrated L2 cache organization of the e600 core is shown in Figure 5-4.

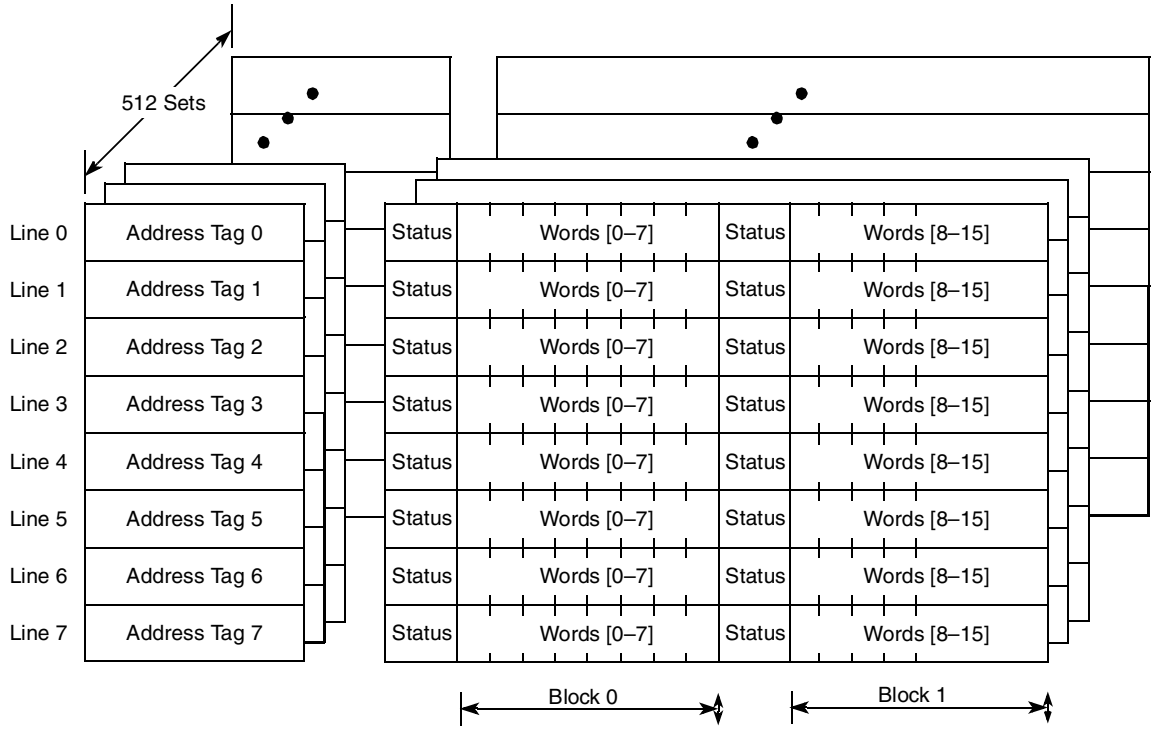


Figure 5-4. L2 Cache Organization

The L2 cache controller contains the L2 cache control register (L2CR), which does the following:

- Includes bits for enabling parity checking on the L2
- Provides for instruction-only and data-only modes
- Provides hardware flushing for the L2
- Selects between two available replacement algorithms for the L2 cache

The L2 implements the MESI cache coherency protocol using three status bits per sector.

Requests from the L1 cache generally result from instruction misses, data load or store misses, write-through operations, or cache management instructions. Requests from the L1 cache are compared against the L2 tags and serviced by the L2 cache if they hit; if they miss in the L2 cache, they go to main memory.

The L2 cache tags are fully pipelined and non-blocking for efficient operation. Thus the L2 cache can be accessed internally while a load for a miss is pending (allowing hits under misses). A reload for a cache miss is treated as a normal access and blocks other accesses for only 1 cycle.

For more information, see the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*.”

5.2.5 Core Interface

The e600 core has an advanced bus interface, the MPX bus interface. The MPX bus includes a 64 data bus and a 36-bit address bus along with sufficient control signals to allow for unique system level optimizations. The MPX bus has the following features:

- Extended 36-bit address bus
- 64-bit data bus; a 32-bit data bus mode is not supported
- Support for a four-state (MESI) cache coherence protocol
- Snooping within the core to maintain L1 data cache, and L2 cache coherency for multiprocessing applications and DMA environments
- Support for address-only transfers (useful for a variety of broadcast operations in multiprocessor applications)
- Address pipelining
- Support for up to 8 out-of-order transactions
- Full data streaming
- Support for data intervention in multiprocessor systems

5.2.6 Overview of Core Interface Accesses

The core interface includes address register queues, prioritization logic, and a bus control unit. The core interface latches snoop addresses for snooping in the L1 data cache and the L2 cache, the memory hierarchy address register queues, and the reservation controlled by the Load Word and Reserve Indexed (**lwarx**) and Store Word Conditional Indexed (**stwcx.**) instructions. Accesses are prioritized with load operations preceding store operations.

Instructions are automatically fetched from the memory system into the instruction unit where they are issued to the execution units at a peak rate of three instructions per clock cycle. Conversely, load and store instructions explicitly specify the movement of operands to and from the integer, floating-point, and AltiVec register files and the memory system.

When the e600 core encounters an instruction or data access, it calculates the effective address and uses the lower-order address bits to check for a hit in the 32-Kbyte L1 instruction and data caches within the core. During L1 cache lookup, the instruction and data memory management units (MMUs) use the higher-order address bits to calculate the virtual address, from which they calculate the physical (real) address. The physical address bits are then compared with the corresponding cache tag bits to determine if a cache hit occurred in the L1 instruction or data cache. If the access misses in the corresponding cache, the transaction is sent to L1 load miss queue or the L1 store miss queue. L1 load miss queue transactions are sent to the internal 256-Kbyte L2 cache. Store miss queue transactions are queued up in the L2 cache controller. If no match is found in the L2 tags, the physical address is used to access system memory.

In addition to loads, stores, and instruction fetches, the e600 core performs hardware table search operations following TLB misses, L1 and L2 cache castout operations, and cache-line snoop push operations when a modified cache line detects a snoop hit from another bus master.

5.2.6.1 Signal Groupings

A subset of the selected internal e600 core signals is grouped as follows:

- Interrupts/resets—These signals include the external interrupt signal, checkstop signals, and both soft reset and hard reset signals. They are used to interrupt and, under various conditions, to reset the core.
- Core status and control—These signals indicate the state of the core. They include the time-base enable, machine quiesce control, and power management signals.
- Clock control—These signals determine the system clock frequency and provide a flexible clocking scheme that allows the processor to operate at an integer multiple of the system clock frequency. They are also used to synchronize multiprocessor systems.
- Test interface—The JTAG (IEEE Std. 1149.1a™-1993) interface and the common on-chip processor (COP) unit provide a serial interface to the core for performing board-level boundary-scan interconnect tests. The test data input (*tdi*) and test data output (*tdo*) scan ports are used to scan instructions as well as data into the various scan registers for JTAG operations. The scan operation is controlled by the test access port (TAP) controller which in turn is controlled by the test mode select (*tms*) input sequence. The scan data is latched in at the rising edge of test clock (*tck*).
- Master address bus—These signals provide information about the type of transfer, such as whether the transaction is bursted, global, write-through, or cache-inhibited. A signal is also provided that indicates if a transfer error occurred.

NOTE

Active-low signals are shown with overbars. For example, $\overline{int}$ (interrupt) and $\overline{sreset}$ (soft reset). 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 *sysclk* (system clock) and *tdo* (JTAG test data output) are referred to as asserted when they are high and negated when they are low.

Figure 5-5 shows the subset of internal core interface signals. Signal functionality is described in detail in the “Core Interface” chapter of the *e600 PowerPC Core Reference Manual*.

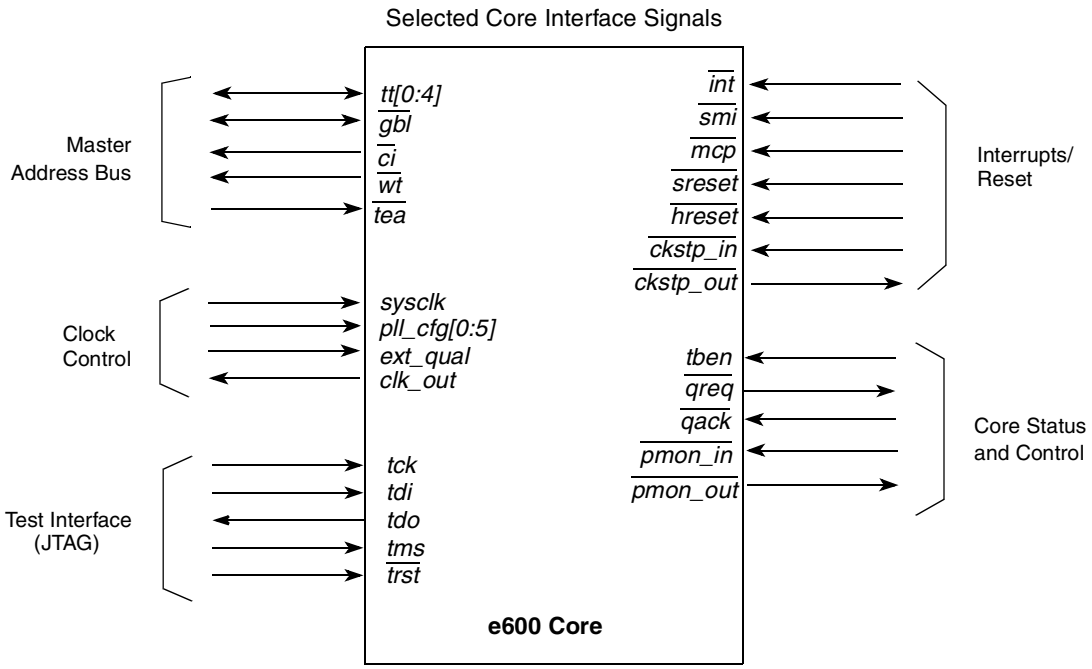


Figure 5-5. Core Interface Signals

5.2.6.2 Clocking

For functional operation, the e600 core uses a single clock input signal, *sysclk*, from which clocking is derived for the processor core and the MPX bus interface. Additionally, internal clock information is made available at the periphery of the core to support debug and development.

The e600 core clocking structure supports a wide range of core-to-MPX bus clock ratios. The internal core clock is synchronized to *sysclk* with the aid of a VCO-based PLL. The *pll_cfg*[0:5] signals are used to program the internal clock rate to a multiple of *sysclk*. The bus clock is maintained at the same frequency as *sysclk*. *sysclk* does not need to be a 50% duty-cycle signal.

5.2.7 Power and Thermal Management

The e600 core is designed for low-power operation. It provides both automatic and program-controlled power reduction modes. If an e600 core functional unit is idle, it automatically goes into a low-power mode. This mode does not affect operational performance. Dynamic power management automatically supplies or withholds power to execution units individually, based upon the contents of the instruction stream. The operation of dynamic power management is transparent to software or any hardware external to the core.

The following two programmable power modes are available to the system:

- Nap—Instruction fetching is halted. Only those clocks for time base, decrementer, and JTAG logic remain running. The e600 core goes into the doze state to snoop memory operations on the MPX bus and then back to nap using a $\overline{qreq}$/$\overline{qack}$ processor-system handshake protocol.
- Sleep—Power consumption is further reduced by disabling bus snooping, leaving only the PLL in a locked and running state. All internal functional units are disabled.

The e600 core also provides an instruction cache throttling mechanism to reduce the instruction execution rate. For thermal management, the e600 core provides a supervisor-level instruction cache throttling control register (ICTC). the “Power and Thermal Management” chapter of the *e600 PowerPC Core Reference Manual* explains how to configure the ICTC register for the e600 core.

5.2.8 Core Performance Monitor

The e600 core incorporates a performance monitor facility that system designers can use to help bring up, debug, and optimize software performance. The performance monitor counts events during execution of instructions related to dispatch, execution, completion, and memory accesses.

The performance monitor incorporates several registers that can be read and written to by supervisor-level software. User-level versions of these registers provide read-only access for user-level applications. These registers are described in Section 5.3.1, “PowerPC ISA Registers and Programming Model.” Performance monitor control registers, MMCR0, MMCR1, and MMCR2 can be used to specify which events are to be counted and the conditions for which a performance monitoring interrupt is taken. Additionally, the sampled instruction address register, SIAR (USIAR), holds the address of the first instruction to complete after the counter overflowed.

Attempting to write to a user-level read-only performance monitor register causes a program interrupt, regardless of the MSR[PR] setting.

When a performance monitor interrupt occurs, program execution continues from vector offset 0x00F00.

The Performance Monitor chapter of the *e600 PowerPC Core Reference Manual* describes the operation of the performance monitor diagnostic tool incorporated in the e600 core.

5.3 e600 Core Architectural Implementation

The PowerPC ISA consists of three layers. Adherence to the PowerPC ISA can be described 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 exception 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 the interrupt model. Implementations that conform to the OEA also adhere to the UISA and the VEA.

The e600 core implementation supports the three levels of the architecture described above. For more information about the PowerPC ISA, see the *Programming Environments Manual for 32-Bit Implementations of the PowerPC Architecture.* Features specific to the e600 core are listed in Section 5.2, "e600 Core Features."

This section describes the PowerPC ISA in general, and specific details about the implementation of the e600 core as a low-power, 32-bit device that implements this architecture. The structure of this section follows the reference manual organization; each subsection provides an overview of that chapter.

- Registers and programming model—Section 5.3.1, "PowerPC ISA Registers and Programming Model," describes the registers for the operating environment architecture common to the e600 core and describes the programming model. It also describes the registers that are unique to the e600 core.

  Instruction set and addressing modes—Section 5.3.2, "Instruction Set," describes the instruction set and addressing modes for the operating environment architecture, and defines and describes the instructions implemented in the e600 core. The information in this section is described more fully in Chapter 6, "e600 Core Registers and Instruction Set Summary."
- Cache implementation—Section 5.3.3, "Cache Implementation within the Core," describes the cache model that is defined generally by the virtual environment architecture. It also provides specific details about the e600 core cache implementation. The information in this section is described more fully in the "L1 and L2 Cache Operation" chapter of the *e600 PowerPC Core Reference Manual*.
- Interrupt model—Section 5.3.4, "Interrupt Model," describes the interrupt model of the operating environment architecture and the differences in the e600 core interrupt model. The information in this section is described more fully in the "Interrupts" chapter of the *e600 PowerPC Core Reference Manual.*
- Memory management—Section 5.3.5, "Memory Management," describes generally the conventions for memory management. This section also describes the e600 core implementation of the 32-bit memory management specification. The information in this section is described more fully in the "Memory Management" chapter of the *e600 PowerPC Core Reference Manual.*
- Instruction timing—Section 5.3.6, "Instruction Timing," provides a general description of the instruction timing provided by the parallel execution supported by the PowerPC ISA and the e600 core. The information in this section is described more fully in the "Instruction Timing" chapter of the *e600 PowerPC Core Reference Manual.*
- AltiVec implementation—Section 5.3.7, "AltiVec Implementation," points out that the e600 core implements AltiVec registers, instructions, and interrupts as described in the *AltiVec Technology Programming Environments Manual.* "AltiVec Technology Implementation" chapter of the *e600 PowerPC Core Reference Manual* provides complete details.

5.3.1 PowerPC ISA Registers and Programming Model

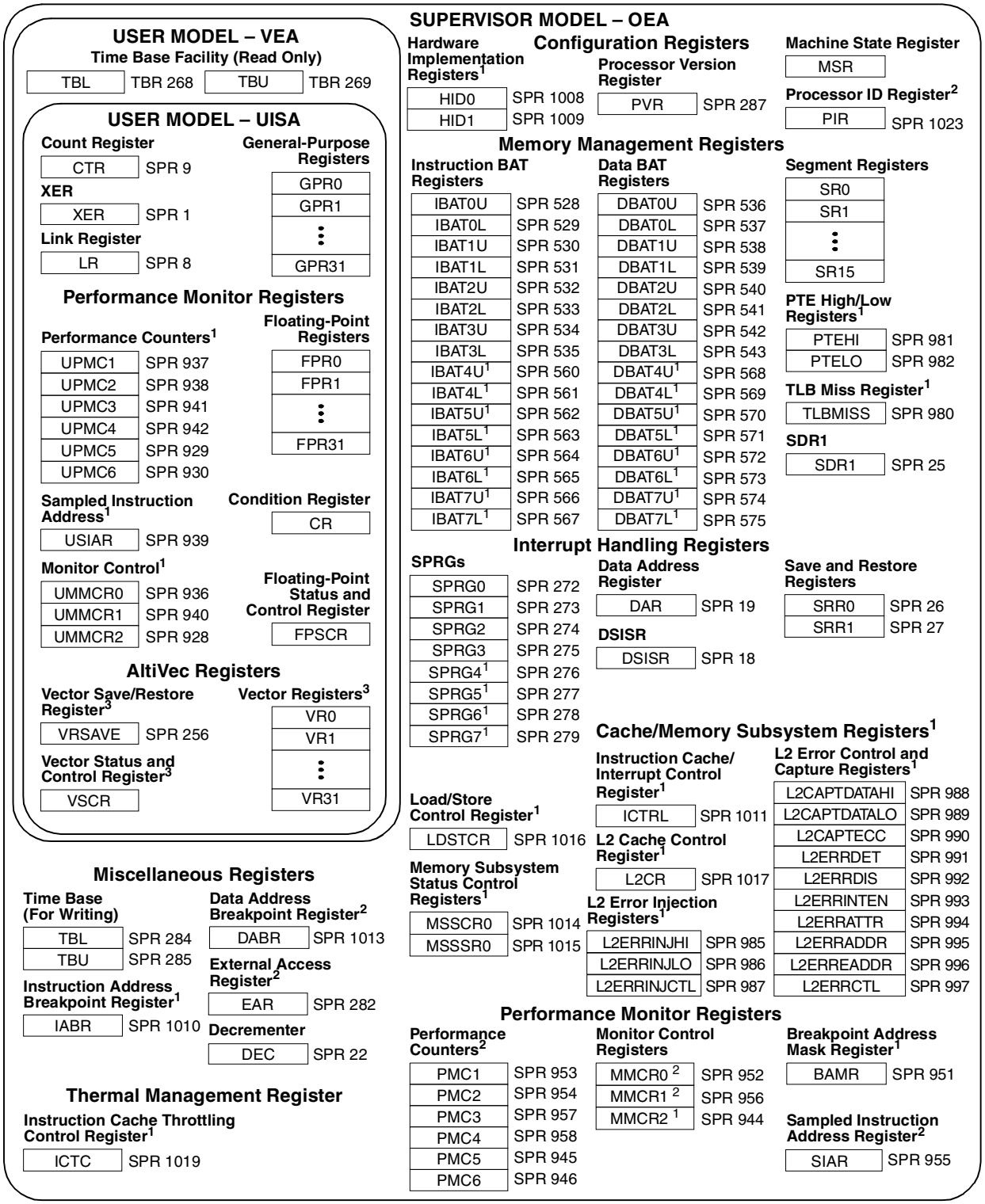
The PowerPC architecture defines register-to-register operations for most computational instructions. Source operands for these instructions are accessed from the registers or are provided as immediate values embedded in the instruction opcode. The three-register instruction format allows specification of a target register distinct from the two source operands. Load and store instructions transfer data between registers and memory.

The PowerPC architecture also defines 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, SPRs, and several miscellaneous registers. The AltiVec extensions to the PowerPC architecture augment the programming model with 32 VRs, one status and control register, and one save and restore register. Each processor that implements the PowerPC architecture also has a unique set of implementation-specific registers to support functionality that may not be defined by the PowerPC architecture.

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 processor, the address translation mechanism, and supervisor registers can be executed only when the processor is operating in supervisor mode.

Figure 5-6 shows all of the e600 core registers, indicating which are available at the user and supervisor levels. The numbers to the right of the SPRs indicate the number that is used in the syntax of the instruction operands to access the register. For more information, see Chapter 6, “e600 Core Registers and Instruction Set Summary.”

The OEA defines numerous SPRs that serve a variety of functions, such as providing controls, indicating status, configuring the processor, and performing special operations. During normal execution, a program can access the registers shown in Figure 5-6, depending on the program’s access privilege (supervisor or user, determined by the privilege-level bit, MSR[PR]). GPRs, FPRs, and VRs 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.



1 e600-specific register that may not be supported on other processors or cores that implement the PowerPC architecture.
2 Register defined as optional in the PowerPC architecture.
3 Register defined by the AltiVec technology.

Figure 5-6. Programming Model—e600 Core Registers

Some registers can be accessed both explicitly and implicitly. In the e600 core, all SPRs are 32 bits wide. Table 6-1 describes the registers implemented by the e600 core.

5.3.2 Instruction Set

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 greatly simplifies instruction pipelining.

For more information, see Chapter 6, "e600 Core Registers and Instruction Set Summary."

5.3.2.1 PowerPC Instruction Set

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 and 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 instructions
  - 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 (**mtspr**, **mfspr**)
  - Move to/from MSR (**mtmsr**, **mfmsr)**
  - Synchronize (**sync**)
  - Instruction synchronize (**isync**)

  - Order loads and stores
- Memory control instructions—These instructions provide control of caches, TLBs, and SRs.
  - Supervisor-level cache management instructions
  - User-level cache instructions
  - Segment register manipulation instructions
  - Translation lookaside buffer management instructions

This grouping 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 four 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 floating-point registers (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.

Processors that implement the PowerPC architecture follow the program flow when they are 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.

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored in 32-bit implementations.

5.3.2.2 AltiVec Instruction Set

The AltiVec instructions are divided into the following categories:

- Vector integer arithmetic instructions—These include arithmetic, logical, compare, rotate, and shift instructions.
- Vector floating-point arithmetic instructions—These include floating-point arithmetic instructions, as well as a discussion on floating-point modes.
- Vector load and store instructions—These include load and store instructions for vector registers. The AltiVec technology defines LRU and transient type instructions that can be used to optimize memory accesses.
  - LRU instructions. The AltiVec architecture specifies that the **lvxl** and **stvxl** instructions differ from other AltiVec load and store instructions in that they leave cache entries in a least-recently-used (LRU) state instead of a most-recently-used state.
  - Transient instructions. The AltiVec architecture describes a difference between static and transient memory accesses. A static memory access should have some reasonable degree of locality and be referenced several times or reused over some reasonably long period of time. A

transient memory reference has poor locality and is likely to be referenced a very few times or over a very short period of time.

The following instructions are interpreted to be transient:

– **dstt** and **dststt** (transient forms of the two data stream touch instructions)
– **lvxl** and **stvxl**

- Vector permutation and formatting instructions—These include pack, unpack, merge, splat, permute, select, and shift instructions.
- Processor control instructions—These instructions are used to read and write from the AltiVec status and control register.
- Memory control instructions—These instructions are used for managing of caches (user level and supervisor level).

5.3.2.3 e600 Core Instruction Set

The e600 core instruction set is defined as follows:

- The e600 core provides hardware support for all 32-bit PowerPC instructions.
- The e600 core implements the following instructions optional to the PowerPC architecture:
  — External Control In Word Indexed (**eciwx**)
  — External Control Out Word Indexed (**ecowx**)
  — Data Cache Block Allocate (**dcba**)
  — Floating Select (**fsel**)
  — Floating Reciprocal Estimate Single-Precision (**fres**)
  — Floating Reciprocal Square Root Estimate (**frsqrte**)
  — Store Floating-Point as Integer Word (**stfiwx**)
  — Load Data TLB Entry (**tlbld**)
  — Load Instruction TLB Entry (**tlbli**)

5.3.3 Cache Implementation within the Core

The following subsections describe the PowerPC architecture's treatment of cache in general, and the e600-specific implementation, respectively. A detailed description of the e600 core cache implementation is provided in the "L1 and L2 Cache Operation" chapter of the *e600 PowerPC Core Reference Manual*.

5.3.3.1 PowerPC Cache Model

The PowerPC architecture does not define hardware aspects of cache implementations. For example, devices that implement the PowerPC architecture can have unified caches, separate L1 instruction and data caches (Harvard architecture), or no cache at all. These devices control the following memory access modes on a page or block basis:

- Write-back/write-through mode
- Caching-inhibited/caching-allowed mode
- Memory coherency required/memory coherency not required mode

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 PowerPC architecture defines the term ‘cache block’ as the cacheable unit. The VEA and OEA define cache management instructions a programmer can use to affect cache contents.

5.3.3.2 e600 Core Cache Implementation

The BPU contains a 128-entry BTIC that provides immediate access to cached target instructions.

5.3.4 Interrupt Model

The following sections describe the PowerPC interrupt model and the e600 core implementation. A detailed description of the e600 core interrupt model is provided in the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual.*

5.3.4.1 PowerPC Interrupt Model

The OEA portion of the PowerPC architecture defines the mechanism by which processors that implement the PowerPC architecture invoke interrupts. Exception conditions may be defined at other levels of the architecture. For example, the UISA defines floating-point exception conditions; the OEA defines the mechanism by which the resulting interrupt is taken.

The PowerPC interrupt mechanism allows the processor to change to supervisor state as a result of unusual conditions arising in the execution of instructions and from signals external to the core, bus errors, or various internal conditions. When interrupts occur, information about the state of the processor is saved to certain registers and the processor begins execution at an address (interrupt vector) predetermined for each interrupt. Processing of interrupts begins in supervisor mode.

Although multiple exception conditions can map to a single interrupt vector, often a more specific condition may be determined by examining a register associated with the interrupt—for example, the DSISR and the floating-point status and control register (FPSCR). Also, software can explicitly enable or disable some exception conditions.

The PowerPC architecture requires that interrupts be taken in program order; therefore, although a particular implementation may recognize exception conditions out of order, they are handled strictly in order with respect to the instruction stream. 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 state, are required to complete before the interrupt is taken. In addition, if a single instruction encounters multiple exception conditions, the resulting interrupts are taken and handled sequentially. Likewise, asynchronous, precise interrupts are recognized when they occur, but are not handled until all instructions currently in the execute stage successfully complete execution and report their results.

To prevent loss of state information, interrupt handlers must save the information stored in the machine status save/restore registers, SRR0 and SRR1, soon after the interrupt is taken to prevent this information from being lost due to another interrupt event. Because exceptions can occur while an interrupt handler routine is executing, multiple interrupts can become nested. It is the interrupt handler’s responsibility to save the necessary state information if control is to return to the interrupted program.

In many cases, after the interrupt handler handles an interrupt, there is an attempt to execute the instruction that caused the interrupt. Instruction execution continues until the next interrupt is encountered. Recognizing and handling interrupt conditions sequentially guarantees that the machine state is recoverable and processing can resume without losing instruction results.

The following terms are used to describe the stages of interrupt processing: recognition, taken, and handling.

- Recognition—Exception recognition occurs when the exception condition that can cause an interrupt is identified by the processor.
- Taken—An interrupt is said to be taken when control of instruction execution is passed to the interrupt handler; that is, the context is saved and the instruction at the appropriate vector offset is fetched and the interrupt handler routine begins executing in supervisor mode.
- Handling—Interrupt handling is performed by the software at the appropriate vector offset. Interrupt handling is begun in supervisor mode.

The occurrence of IEEE Std.754 floating-point exceptions may not cause an exception to be taken. These IEEE-defined exceptions are referred to as IEEE floating-point exceptions or floating-point exceptions.

5.3.4.2 e600 Core Interrupts

As specified by the PowerPC architecture, interrupts can be either precise or imprecise and either synchronous or asynchronous. Asynchronous interrupts are caused by events external to the core's execution; synchronous interrupts are caused by instructions.

The types of interrupts are shown in Table 5-1. Note that all interrupts except for the performance monitor, AltiVec unavailable, instruction address breakpoint, system management, AltiVec assist, and the three software table search interrupts are described in Chapter 6, "Interrupts," in *The Programming Environments Manual*.

Table 5-1. e600 Core Interrupt Classifications

Synchronous/Asynchronous	Precise/Imprecise	Interrupt Types
Asynchronous, nonmaskable	Imprecise	System reset, machine check
Asynchronous, maskable	Precise	External interrupt, system management interrupt, decrementer interrupt, performance monitor interrupt
Synchronous	Precise	Instruction-caused interrupts

The interrupt classifications are discussed in greater detail in the "Interrupts" chapter of the *e600 PowerPC Core Reference Manual* For a better understanding of how the e600 core implements precise interrupts, see the "Instruction Timing" chapter of the *e600 PowerPC Core Reference Manual.* Table 5-2 lists the interrupts implemented in the e600 core and conditions that cause them. Table 5-2 also notes the e600-specific interrupts.

The three software table search interrupts support software page table searching and are enabled by setting HID0[STEN]. See the "Interrupts" and "Memory Management" chapters of the *e600 PowerPC Core Reference Manual.*

Table 5-2. Interrupts and Exception Conditions

Interrupt Type	Vector Offset	Causing Conditions
Reserved	0x00000	—
System reset	0x00100	Assertion of either *hreset* or *sreset* or at power-on reset
Machine check	0x00200	Assertion of *tea* during a data bus transaction (see Section 5.3.4.2.1, "Sources of tea_ assertion," for more information), assertion of *mcp*, an address bus parity error on the MPX bus, a data bus parity error on the MPX bus, an L1 instruction cache error, an L1 data cache error, and a core memory subsystem detected error including the following: • L2 data parity error • L2 tag parity error • Single-bit and multiple-bit L2 ECC errors MSR[ME] must be set.
DSI	0x00300	As specified in the PowerPC architecture. Also includes the following: • A hardware table search due to a TLB miss on load, store, or cache operations results in a page fault • Any load or store to a direct-store segment (SR[T] = 1) • A **lwarx** or **stwcx.** instruction to memory with cache-inhibited or write-through memory/cache access attributes.
ISI	0x00400	As specified in the PowerPC architecture
External interrupt	0x00500	MSR[EE] = 1 and *int* is asserted
Alignment	0x00600	• A floating-point load/store, **stmw**, **stwcx.**, **lmw**, **lwarx**, **eciwx**, or **ecowx** instruction operand is not word-aligned. • A multiple/string load/store operation is attempted in little-endian mode • An operand of a **dcbz** instruction is on a page that is write-through or cache-inhibited for a virtual mode access. • An attempt to execute a **dcbz** instruction occurs when the cache is disabled or locked.
Program	0x00700	As specified in the PowerPC architecture
Floating-point unavailable	0x00800	As specified in the PowerPC architecture
Decrementer	0x00900	As defined by the PowerPC architecture, when the msb of the DEC register changes from 0 to 1 and MSR[EE] = 1
Reserved	0x00A00–00BFF	—
System call	0x00C00	Execution of the System Call (**sc**) instruction
Trace	0x00D00	MSR[SE] =1 or a branch instruction is completing and MSR[BE] =1. The e600 core operates as specified in the OEA by taking this interrupt on an **isync**.

Table 5-2. Interrupts and Exception Conditions (continued)

Interrupt Type	Vector Offset	Causing Conditions
Reserved	0x00E00	The e600 core does not generate an interrupt to this vector. Other processors may use this vector for floating-point assist interrupts.
Reserved	0x00E10–00EFF	—
Performance monitor	0x00F00	The limit specified in PMC*n* is met and MMCR0[ENINT] = 1 (e600-specific)
AltiVec unavailable	0x00F20	Occurs due to an attempt to execute any non-streaming AltiVec instruction when MSR[VEC] = 0. This interrupt is not taken for data streaming instructions (**dst*x***, **dss,** or **dssall**). (e600-specific)
ITLB miss	0x01000	Caused when HID0[STEN] = 1 and the effective address for an instruction fetch cannot be translated by the ITLB (e600-specific).
DTLB miss-on-load	0x01100	Caused when HID0[STEN] = 1 and the effective address for a data load operation cannot be translated by the DTLB (e600-specific).
DTLB miss-on-store	0x01200	Caused when HID0[STEN] = 1, and either the effective address for a data store operation cannot be translated by the DTLB, or when a DTLB hit occurs, and the changed bit in the PTE must be set due to a data store operation (e600-specific).
Instruction address breakpoint	0x01300	IABR[0–29] matches EA[0–29] of the next instruction to complete and IABR[BE] = 1 (e600-specific).
System management interrupt	0x01400	MSR[EE] = 1 and $\overline{smi}$ are asserted (e600-specific). See section, “System Management Interrupt (0x01400)” of the “Interrupts” chapter of the e600 PowerPC Core Reference Manual.
Reserved	0x01500–015FF	—
AltiVec assist	0x01600	This e600-specific interrupt supports denormalization detection in Java™ mode as specified in the *AltiVec Technology Programming Environments Manual* in Chapter 3, “Operand Conventions.”
Reserved	0x01700–02FFF	—

5.3.4.2.1 Sources of *tea_* assertion

When the core performs a read transaction to a target on one of the external interfaces, a bus error may occur, which in turn causes the assertion of *tea_* to the core. Table 5-3 summarizes the potential sources of *tea_* assertion and the associated error reporting register fields.

Table 5-3. *tea_* Sources

Read Target	Error Register	Error Register Fields
MCM	EDR	LAE
DDR	ERR_DETECT	MBE MSE
eLBC	LTESR	DM PAR

Table 5-3. *tea_* Sources (continued)

Read Target	Error Register	Error Register Fields
PCI	ERR_DR	Addr Parity error Rcvd SERR error Mstr PERR error Trgt PERR error Mstr abort error Trgt abort error ORMSV error SCM error TOE error
PCI Express	PEX_ERR_DR	PCT PCAC CDNSC CRSNC IACA CRST IOIS CIS CIEP IOIEP OAC IOIA
	PEX_PME_MES_DR	HRD LDD

5.3.5 Memory Management

The following subsections describe the memory management features of the PowerPC architecture, and the e600 core implementation, respectively.

5.3.5.1 PowerPC Memory Management Model

The primary function of the MMU in a processor or core that implements the PowerPC architecture is the translation of logical (effective) addresses to physical addresses (referred to as real addresses in the architecture specification) for memory accesses and I/O accesses (I/O accesses are assumed to be memory-mapped). In addition, the MMU provides access protection on a segment, block, or page basis. Note that the e600 core does not implement the optional direct-store facility.

Two general types of memory accesses generated by processors that implement the PowerPC architecture require address translation—instruction accesses and data accesses generated by load and store instructions. In addition, the addresses specified by cache instructions and the optional external control instructions also require translation. Generally, the address translation mechanism is defined in terms of the segment descriptors and page tables that the processors use to locate the effective-to-physical address mapping for memory accesses. The segment information translates the effective address to an interim virtual address, and the page table information translates the virtual address to a physical address.

The segment descriptors, used to generate the interim virtual addresses, are stored as segment registers on 32-bit implementations (such as the e600 core). In addition, two translation lookaside buffers (TLBs) are

implemented on the e600 core to keep recently used page address translations within the core. Although the PowerPC OEA describes one MMU (conceptually), the e600 core hardware maintains separate TLBs and table search resources for instruction and data accesses that can be performed independently (and simultaneously). Therefore, the e600 core is described as having two MMUs, one for instruction accesses (IMMU) and one for data accesses (DMMU).

The block address translation (BAT) mechanism is a software-controlled array that stores the available block address translations within the core. BAT array entries are implemented as pairs of BAT registers that are accessible as supervisor special-purpose registers (SPRs). There are separate instruction and data BAT mechanisms. In the e600 core, they reside in the instruction and data MMUs, respectively.

The MMUs, together with the interrupt processing mechanism, provide the necessary support for the operating system to implement a paged virtual memory environment and for enforcing protection of designated memory areas. The “Interrupts” chapter of the *e600 PowerPC Core Reference Manual* describes how the MSR controls critical MMU functionality.

5.3.5.2 e600 Core Memory Management Implementation

The e600 core implements separate MMUs for instructions and data. It maintains a copy of the segment registers in the instruction MMU; however, read and write accesses to the segment registers (**mfsr** and **mtsr**) are handled through the segment registers in the data MMU. The e600 core MMU is described in the “Memory Management” chapter of the *e600 PowerPC Core Reference Manual.*

The e600 core implements the memory management specification of the PowerPC OEA for 32-bit implementations but adds capability for supporting 36-bit physical addressing. Thus, it provides 4 Gbytes of physical address space accessible to supervisor and user programs, with a 4-Kbyte page size and 256-Mbyte segment size. In addition, the e600 core MMUs use an interim virtual address (52 bits) and hashed page tables in the generation of 32- or 36-bit physical addresses (depending on the setting of HID0[XAEN]). Processors that implement the PowerPC architecture also have a BAT mechanism for mapping large blocks of memory. Block range from 128 Kbytes to 256 Mbytes and are software programmable.

The e600 core provides table search operations performed in hardware. The 52-bit virtual address is formed and the MMU attempts to fetch the PTE that contains the physical address from the appropriate TLB within the core. If the translation is not found in either the BAT array or in a TLB (that is, a TLB miss occurs), the hardware performs a table search operation (using a hashing function) to search for the PTE. Hardware table searching is the default mode for the e600 core; however, if HID0[STEN] = 1, a software table search is performed.

The e600 core also provides support for table search operations performed in software (if HID0[STEN] is set). In this case, the TLBMISS register saves the effective address of the access that requires a software table search. The PTEHI and PTELO registers and the **tlbli** and **tlbld** instructions are used in reloading the TLBs during a software table search operation.

The following interrupts support software table searching if HID0[STEN] is set and a TLB miss occurs:

- For an instruction fetch, an ITLB miss interrupt
- For a data load, a DTLB miss-on-load interrupt
- For a data store, a DTLB miss-on-store interrupt

The e600 core implements the optional TLB invalidate entry (**tlbie**) and TLB synchronize (**tlbsync**) instructions that can be used to invalidate TLB entries.

5.3.6 Instruction Timing

This section describes how the e600 core performs operations defined by instructions and reports the results of instruction execution. The e600 core design minimizes average instruction execution latency, which is the number of clock cycles it takes to fetch, decode, dispatch, issue, and execute instructions and make results available for subsequent instructions. Some instructions, such as loads and stores, access memory and require additional clock cycles between the execute phase and the write-back phase. Latencies depend on whether an access is to cacheable or noncacheable memory, whether it hits in the L1 or L2 cache, whether a cache access generates a write back to memory, whether the access causes a snoop hit from another device that generates additional activity, and other conditions that affect memory accesses.

To improve throughput, the e600 core implements pipelining, superscalar instruction issue, branch folding, removal of fall-through branches, three-level speculative branch handling, and multiple execution units that operate independently and in parallel.

As an instruction passes from stage to stage, the subsequent instruction can follow through the stages as the preceding instruction vacates them, allowing several instructions to be processed simultaneously. Although it may take several cycles for an instruction to pass through all the stages, when the pipeline is full, one instruction can complete its work on every clock cycle. Figure 5-7 represents a generic four-stage pipelined execution unit, which when filled has a throughput of one instruction per clock cycle.

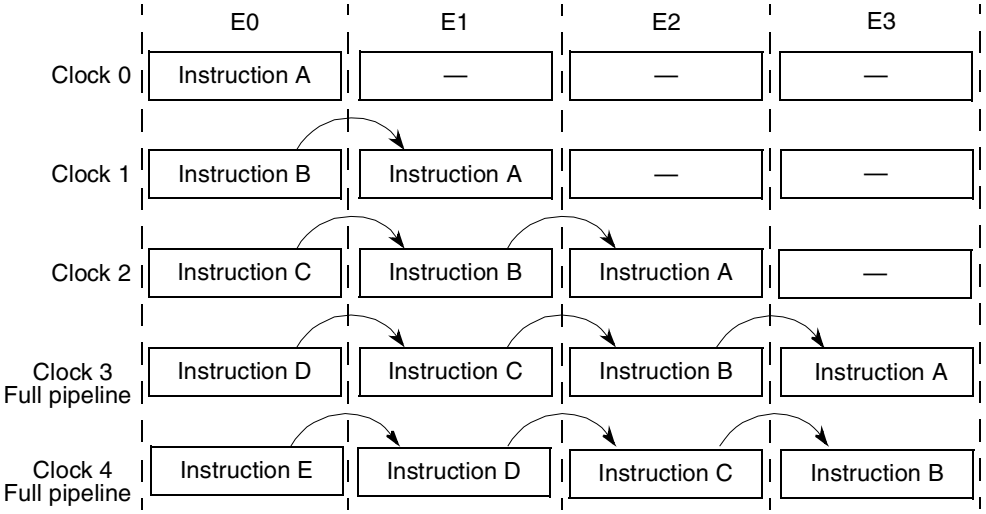


Figure 5-7. Pipelined Execution Unit

Figure 5-8 shows the entire path that instructions take through the fetch1, fetch2, decode/dispatch, execute, issue, complete, and write-back stages, which is considered the master pipeline of the e600 core. The FPU, LSU, IU2, VIU2, VFPU, and VPU are multiple-stage pipelines.

The e600 core contains the following execution units:

- Branch processing unit (BPU)

- Three integer unit 1s (IU1a, IU1b, and IU1c)—execute all integer instructions except multiply, divide, and move to/from SPR instructions.
- Integer unit 2 (IU2)—executes miscellaneous instructions including the CR logical operations, integer multiplication and division instructions, and move to/from special-purpose register instructions
- 64-bit floating-point unit (FPU)
- Load/store unit (LSU)
- The AltiVec unit contains the following four independent execution units for vector computations:
  - AltiVec permute unit (VPU)
  - AltiVec integer unit 1 (VIU1)
  - Vector integer unit 2 (VIU2)
  - Vector floating-point unit (VFPU)

  A maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). An instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability. Moreover, the VIU2, VFPU, and VPU are pipelined, so they can operate on multiple instructions.

The e600 core can complete as many as three instructions on each clock cycle. In general, the e600 core processes instructions in seven stages—fetch1, fetch2, decode/dispatch, issue, execute, complete, and write-back, as shown in Figure 5-8.

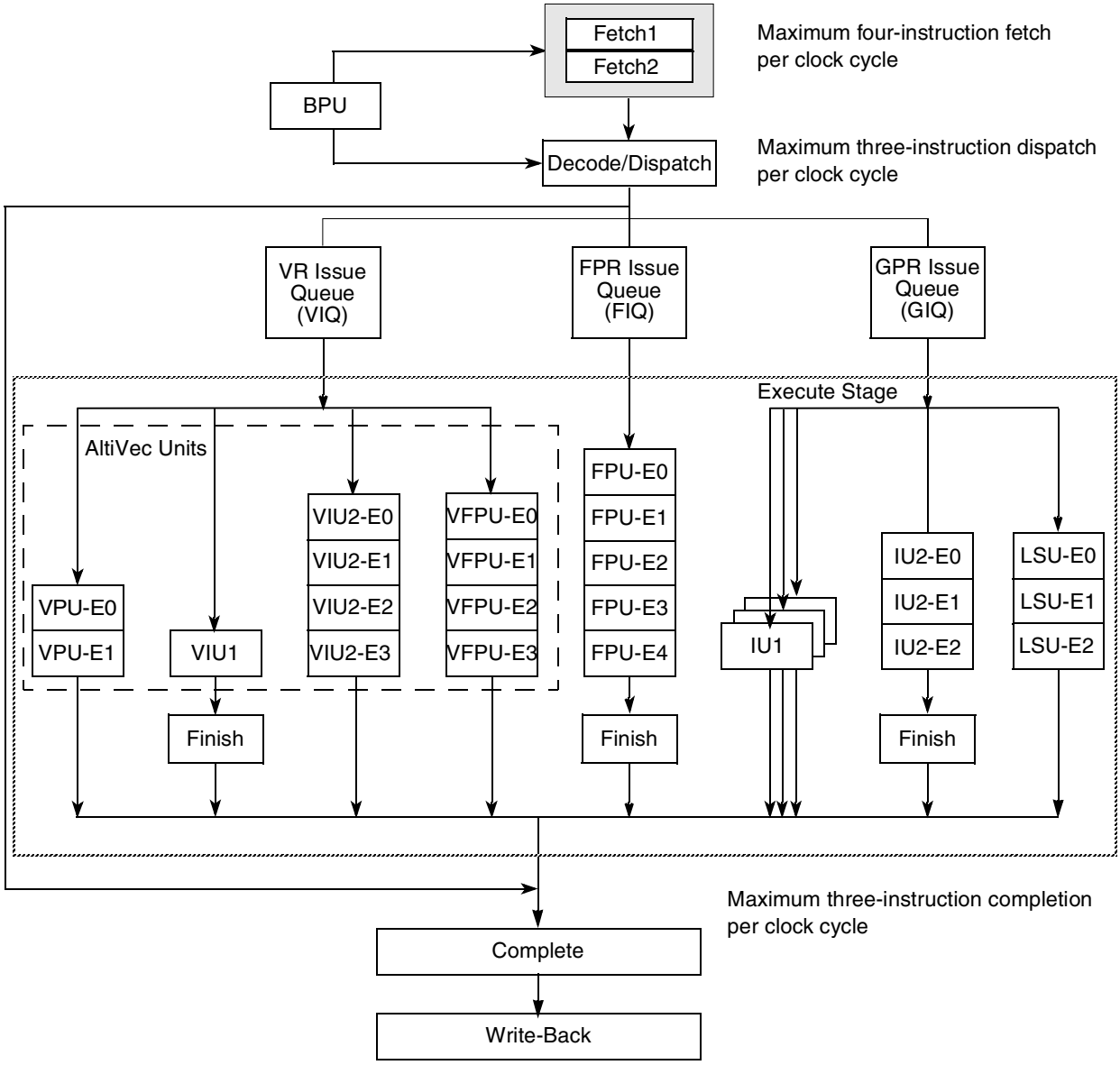


Figure 5-8. Superscalar/Pipeline Diagram

The instruction pipeline stages are described as follows:

- Instruction fetch—Includes the clock cycles necessary to request an instruction and the time the memory system takes to respond to the request. Instructions retrieved are latched into the instruction queue (IQ) for subsequent consideration by the dispatcher.

  Instruction fetch timing depends on many variables, such as whether an instruction is in the branch target instruction cache (BTIC), the instruction cache, or the L2 cache. Those factors increase when it is necessary to fetch instructions from system memory and include the core-to-MPX bus clock ratio, the amount of bus traffic, and whether any cache coherency operations are required.

- The decode/dispatch stage fully decodes each instruction; most instructions are dispatched to the issue queues (branch, **isync**, **rfi**, and **sc** instructions do not go to issue queues).
- The three issue queues, FIQ, VIQ, and GIQ, can accept as many as one, two, and three instructions, respectively, in a cycle. Instruction dispatch requires the following:
  - Instructions are dispatched only from the three lowest IQ entries—IQ0, IQ1, and IQ2.
  - A maximum of three instructions can be dispatched to the issue queues per clock cycle.
  - Space must be available in the CQ for an instruction to dispatch (this includes instructions that are assigned a space in the CQ but not an issue queue).
- The issue stage reads source operands from rename registers and register files and determines when instructions are latched into the execution unit reservation stations. The GIQ, FIQ, and VIQ (AltiVec) issue queues have the following similarities:
  - Operand lookup in the GPRs, FPRs, and VRs, and their rename registers.
  - Issue queues issue instructions to the proper execution units.
  - Each issue queue holds twice as many instructions as can be dispatched to it in one cycle; the GIQ has six entries, the VIQ has four, and the FIQ has two.

  The three issue queues are described as follows:
  - The GIQ accepts as many as three instructions from the dispatch unit each cycle. IU1, IU2, and all LSU instructions (including floating-point and AltiVec loads and stores) are dispatched to the GIQ.
  - Instructions can be issued out-of-order from the bottom three GIQ entries (GIQ2–GIQ0). An instruction in GIQ1 destined for an IU1 does not have to wait for an instruction in GIQ0 that is stalled behind a long-latency integer divide instruction in the IU2.
  - The VIQ accepts as many as two instructions from the dispatch unit each cycle. All AltiVec instructions (other than load, store, and vector touch instructions) are dispatched to the VIQ. A maximum of two AltiVec instructions can be issued out-of-order to any combination of AltiVec execution units per clock cycle from the bottom two VIQ entries (VIQ1–VIQ0). This means an instruction in VIQ1 does not have to wait for an instruction in VIQ0 that is waiting for operand availability.
  - The FIQ can accept one instruction from the dispatch unit per clock cycle. It looks at the first instruction in its queue and determines if the instruction can be issued to the FPU in this cycle.
- The execute stage accepts instructions from its issue queue when the appropriate reservation stations are not busy. In this stage, the operands assigned to the execution stage from the issue stage are latched.

  The execution unit executes the instruction (perhaps over multiple cycles), writes results on its result bus, and notifies the CQ when the instruction finishes. The execution unit reports any interrupts to the completion stage. Instruction-generated interrupts are not taken until the excepting instruction is next to retire.

  Most integer instructions have a 1-cycle latency, so results of these instructions are available 1 clock cycle after an instruction enters the execution unit. The FPU, LSU, IU2, VIU2, VFPU, and VPU units are pipelined, as shown in the “AltiVec Technology Implementation” chapter of the *e600 PowerPC Core Reference Manual*.

Note that AltiVec computational instructions are executed in the four independent, pipelined AltiVec execution units. The VPU has a two-stage pipeline, the VIU1 has a one-stage pipeline, and the VIU2 and VFPU have four-stage pipelines. As many as 10 AltiVec instructions can be executing concurrently.

- The complete and write-back stages maintain the correct architectural machine state and commit results to the architected registers in the proper order. If completion logic detects an instruction containing an interrupt status, all following instructions are cancelled, their execution results in the rename buffers are discarded, and the correct instruction stream is fetched.

  The complete stage ends when the instruction is retired. Three instructions can be retired per clock cycle. If no dependencies exist, as many as three instructions are retired in program order. The "Instruction Timing" chapter of the *e600 PowerPC Core Reference Manual* describes completion dependencies.

The write-back stage occurs in the clock cycle after the instruction is retired.

5.3.7 AltiVec Implementation

The e600 core implements the AltiVec registers and instruction set as they are described in the *AltiVec Technology Programming Environments Manual* in Chapter 2, "AltiVec Register Set," and in Chapter 6, "AltiVec Instructions." One additional implementation-specific interrupt, the AltiVec assist interrupt, is used in handling denormalized numbers in Java mode.

Both interrupts are described fully in the "Interrupts" chapter of the *e600 PowerPC Core Reference Manual.* Also, the default setting for the VSCR[NJ] bit is Java–compliant (VSCR[NJ] = 0) in the e600 core. The AltiVec implementation is described fully in the "AltiVec Technology Implementation" chapter of the *e600 PowerPC Core Reference Manual*.

Chapter 6
e600 Core Registers and Instruction Set Summary

This chapter describes the PowerPC ISA registers on the e600 core, emphasizing those features specific to the e600 core and summarizing those that are common to processors implementing the PowerPC ISA. It consists of three major sections, which describe the following:

- Registers implemented in the e600 core
- Operand conventions
- The e600 core instruction set

For detailed information about architecture-defined features, see the *Programming Environments Manual* and the *AltiVec Technology Programming Environments Manual*.

AltiVec Technology and the Programming Model

AltiVec programming model features are described as follows:

- Thirty-four additional registers—32 VRs, VRSAVE, and VSCR. See the “AltiVec Technology Implementation” chapter of the *e600 PowerPC Core Reference Manual*.

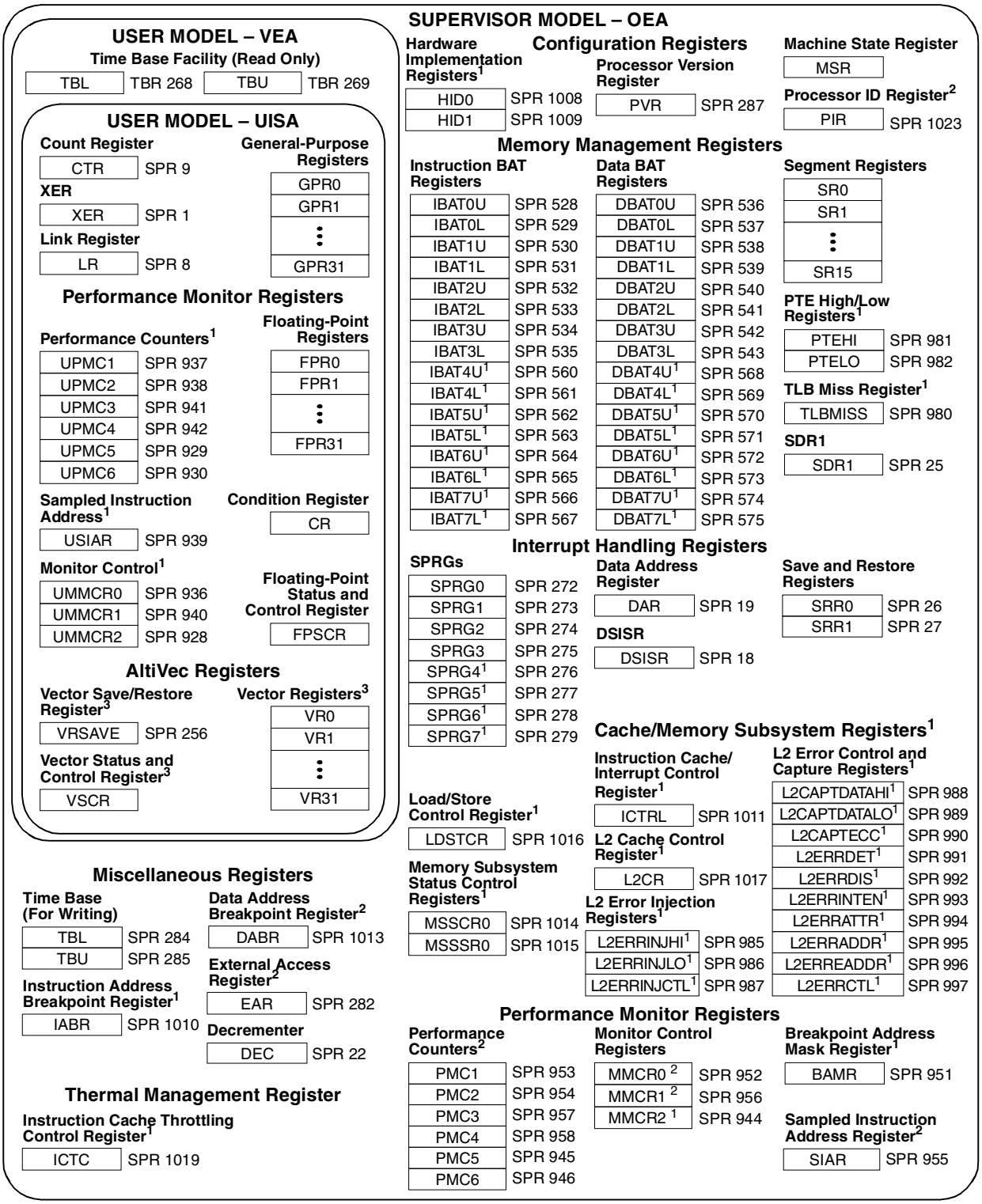
6.1 e600 Core Register Set

This section describes the registers implemented in the e600 core. It includes an overview of registers defined by the PowerPC ISA and the AltiVec technology, highlighting differences in how these registers are implemented in the e600 core, and a detailed description of e600-core-specific registers. Full descriptions of the architecture-defined register set are provided in Chapter 2, “Register Set,” in the *Programming Environments Manual*, Chapter 2, “AltiVec Register Set,” in the *AltiVec Technology Programming Environments Manual* (PEM), and in the *e600 PowerPC Core Reference Manual* (e600RM).

Registers are defined at all three levels of the PowerPC ISA—user instruction set architecture (UISA), virtual environment architecture (VEA), and operating environment architecture (OEA). The PowerPC ISA defines register-to-register operations for all computational instructions. Source data for these instructions is accessed from the registers within the core or provided as immediate values embedded in the opcode. The three-register instruction format allows specification of a target register distinct from the two source registers, thus preserving the original data for use by other instructions and reducing the number of instructions required for certain operations. Data is transferred between memory and registers with explicit load and store instructions only.

6.1.1 Register Set Overview

Figure 6-1 shows the e600 core register set.



1 e600-specific register that may not be supported on other processors that implement the PowerPC architecture.
2 Register defined as optional in the PowerPC architecture.
3 Register defined by the AltiVec technology.

Figure 6-1. Programming Model—e600 Core Registers

The number to the right of the special-purpose registers (SPRs) is the number used in the syntax of the instruction operands to access the register (for example, the number used to access the XER register is SPR 1). These registers can be accessed using **mtspr** and **mfspr**. Note that not all registers in Figure 6-1 are SPRs; for example, VSCR and VRs are AltiVec registers and do not have an SPR number.

6.1.2 e600 Core Register Set

Table 6-1 summarizes the registers implemented in the e600 core.

6.1.3 User-Level Registers (UISA)

Table 6-1. e600 Core Register Summary

Name	SPR	Description	Reference/ Section
UISA Registers			
CR	—	Condition register. The 32-bit CR consists of eight 4-bit fields, CR0–CR7, that reflect results of certain arithmetic operations and provide a mechanism for testing and branching.	PEM
CTR	9	Count register. Holds a loop count that can be decremented during execution of appropriately coded branch instructions. The CTR can also provide the branch target address for the Branch Conditional to Count Register (**bcctr***x*) instruction.	PEM
FPR0–FPR31	—	Floating-point registers (FPR*n*). The 32 FPRs serve as the data source or destination for all floating-point instructions.	PEM
FPSCR	—	Floating-point status and control register. Contains floating-point exception signal bits, exception summary bits, exception enable bits, and rounding control bits for compliance with the IEEE 754 standard.	PEM
GPR0–GPR31	—	General-purpose registers (GPR*n*). The thirty-two GPRs serve as data source or destination registers for integer instructions and provide data for generating addresses.	PEM
LR	8	Link register. Provides the branch target address for the Branch Conditional to Link Register (**bclr***x*) instruction, and can be used to hold the logical address of the instruction that follows a branch and link instruction, typically used for linking to subroutines.	PEM
UMMCR0 [1] UMMCR1 [1] UMMCR2 [1]	936 940 928	User monitor mode control registers (UMMCR*n*). Used to enable various performance monitor interrupt functions. UMMCRs provide user-level read access to MMCR registers.	6.1.6.9.2, e600RM, 6.1.6.9.4, 6.1.6.9.6,
UPMC1–UPMC6 [1]	937, 938 941, 942 929, 930	User performance monitor counter registers (UPMC*n*). Used to record the number of times a certain event has occurred. UPMCs provide user-level read access to PMC registers.	6.1.6.9.9, e600RM
USIAR [1]	939	User sampled instruction address register. Contains the effective address of an instruction executing at or around the time that the core signals the performance monitor interrupt. USIAR provides user-level read access to the SIAR.	6.1.6.9.11 e600RM
VR0–VR31 [2]	—	Vector registers (VR*n*). Data source and destination registers for all AltiVec instructions.	e600RM
VRSAVE [2]	256	Vector save/restore register. Defined by the AltiVec technology to assist application and operating system software in saving and restoring the architectural state across process context-switched events. The register is maintained only by software to track live or dead information on each AltiVec register.	e600RM
VSCR [2]	—	Vector status and control register. A 32-bit vector register that is read and written in a manner similar to the FPSCR.	e600RM
XER	1	Indicates overflows and carries for integer operations. **Implementation Note**—To emulate the **lscbx** instruction, XER[16–23] are be read with **mfspr**[XER] and written with **mtspr**[XER].	PEM

Table 6-1. e600 Core Register Summary (continued)

Name	SPR	Description	Reference/ Section
VEA			
TBL, TBU (For reading)	TBR 268 TBR 269	Time base facility. Consists of two 32-bit registers, time base lower and upper registers (TBL/TBU). TBL (TBR 268) and TBU (TBR 269) can only be read from and not written to.TBU and TBL can be read with the move from time base register (**mftb**) instruction. **Implementation Note**—Reading from SPR 284 or 285 using the **mftb** instruction causes an illegal instruction interrupt.	PEM, 6.1.5.1, 6.3.5.1
OEA			
BAMR [1]	951	Breakpoint address mask register. Used in conjunction with the events that monitor IABR hits. **Note:** See Table 6-37 for specific synchronization requirements on this register.	6.1.6.9.7 e600RM
DABR [3]	1013	Data address breakpoint register. Optional register implemented in the e600 core and is used to cause a breakpoint interrupt if a specified data address is encountered. See Table 6-37 for specific synchronization requirements on this register.	PEM
DAR	19	Data address register. After a DSI or alignment interrupt, DAR is set to the effective address (EA) generated by the faulting instruction.	PEM
DEC	22	Decrementer register. A 32-bit decrementer counter used with the decrementer interrupt. **Implementation Note**—In the e600 core, DEC is decremented and the time base increments at 1/4 of the MPX bus clock frequency.	PEM
DSISR	18	DSI source register. Defines the cause of DSI and alignment interrupts.	PEM
EAR	282	External access register. Used with **eciwx** and **ecowx**. Note that the EAR and the **eciwx** and **ecowx** instructions are optional in the PowerPC ISA. Since **eciwx** and **ecowx** are not implemented in the e600 core, the EAR must not be enabled. Otherwise, attempted execution of **eciwx**/**ecowx** causes boundedly undefined results. See Table 6-37 for specific synchronization requirements on this register.	PEM
HID0 [1] HID1 [1]	1008, 1009	Hardware implementation-dependent registers. Control various functions, such as the power management features, and locking, enabling, and invalidating the instruction and data caches. The HID1 includes bits that reflects the state of *pll_cfg*[0:5] clock signals and control other bus-related functions. See Table 6-37 for specific synchronization requirements on this register.	6.1.6.1, 6.1.6.2
IABR [1]	1010	Instruction address breakpoint register. Used to cause a breakpoint interrupt if a specified instruction address is encountered. See Table 6-37 for specific synchronization requirements on this register.	6.1.6.6

Table 6-1. e600 Core Register Summary (continued)

Name	SPR	Description	Reference/ Section
IBAT0U/L IBAT1U/L IBAT2U/L IBAT3U/L IBAT4U/L [1] IBAT5U/L [1] IBAT6U/L [1] IBAT7U/L [1] DBAT0U/L DBAT1U/L DBAT2U/L DBAT3U/L DBAT4U/L [1] DBAT5U/L [1] DBAT6U/L [1] DBAT7U/L [1]	528, 529 530, 531 532, 533 534, 535 560, 561 562, 563 564, 565 566, 567 536, 537 538, 539 540, 541 542, 543 568, 569 570, 571 572, 573 574, 575	Block-address translation (BAT) registers. The OEA includes an array of block address translation registers that can be used to specify four blocks of instruction space and four blocks of data space. The BAT registers are implemented in pairs: four pairs of instruction BATs (IBAT0U–IBAT3U and IBAT0L–IBAT3L) and four pairs of data BATs (DBAT0U–DBAT3U and DBAT0L–DBAT3L). Sixteen additional BAT registers are available for the e600 core. These registers are enabled by setting HID0[HIGH_BAT_EN]. When HID0[HIGH_BAT_EN] = 1, the 16 additional BAT registers, organized as 4 pairs of instruction BAT registers (IBAT4U–IBAT7U paired with IBAT4L–IBAT7L) and 4 pairs of data BAT registers (DBAT4U–DBAT7U paired with DBAT4L–DBAT7L) are available. Thus, the e600 can define a total of 16 blocks implemented as 32 BAT registers. Because BAT upper and lower words are loaded separately, software must ensure that BAT translations are correct during the time that both BAT entries are being loaded. The e600 core implements IBAT[G]; however, attempting to execute code from an IBAT area with G = 1 causes an ISI interrupt. See Table 6-37 for specific synchronization requirements on this register.	PEM e600RM
ICTC [1]	1019	Instruction cache throttling control register. Has bits for enabling instruction cache throttling and for controlling the interval at which instructions are fetched. This controls overall junction temperature.	6.1.6.8 e600RM
ICTRL [1]	1011	Instruction cache and interrupt control register. Used in configuring interrupts and error reporting for the instruction and data caches. See Table 6-37 for specific synchronization requirements on this register.	6.1.6.5.15
L2CR [1]	1017	L2 cache control register. Includes bits for enabling parity checking, setting the L2 cache size, and flushing and invalidating the L2 cache.	6.1.6.5.1
L2ERRINJHI [1] L2ERRINJLO [1] L2ERRINJCTL [1] L2CAPTDATAHI [1] L2CAPTDATALO [1] L2CAPTDATAECC [1] L2ERRDET [1] L2ERRDIS [1] L2ERRINTEN [1] L2ERRATTR [1] L2ERRADDR [1] L2ERREADDR [1] L2ERRCTL [1]	985 986 987 988 989 990 991 992 993 994 995 996 997	L2 error registers. The L2 cache supports error injection into the L2 data, data ECC or tag, which can be used to test error recovery software by deterministically creating error scenarios. L2ERRINJHI, L2ERRINJLO, and L2ERRINJCTL are error injection registers. The error control and capture registers control the detection and reporting of tag parity and ECC errors.	6.1.6.5.2 6.1.6.5.3 6.1.6.5.4 6.1.6.5.5 6.1.6.5.6 6.1.6.5.7 6.1.6.5.8 6.1.6.5.9 6.1.6.5.10 6.1.6.5.11 6.1.6.5.12 6.1.6.5.13 6.1.6.5.14
LDSTCR [1]	1016	Load/store control register. Controls data L1 cache way-locking. See Table 6-37 for specific synchronization requirements on this register.	6.1.6.5.16
MMCR0 [3] MMCR1 [3] MMCR2 [1]	952 956 944	Monitor mode control registers (MMCR*n*). Enable various performance monitor interrupt functions. UMMCR0–UMMCR2 provide user-level read access to these registers.	6.1.6.9.1 6.1.6.9.3 6.1.6.9.5

Table 6-1. e600 Core Register Summary (continued)

<table>
<tr><th>Name</th><th>SPR</th><th>Description</th><th>Reference/ Section</th></tr>
<tr><td>MSR</td><td>—</td><td>Machine state register. Defines the processor state. The MSR can be modified by the mtmsr, sc, and rfi instructions. It can be read by the mfmsr instruction. When an interrupt is taken, MSR contents are saved to SRR1. See the “Interrupts” chapter of the e600 PowerPC Core Reference Manual. The following bits are optional in the PowerPC ISA.
Note that setting MSR[EE] masks decrementer and external interrupts and e600-core-specific system management and performance monitor interrupts.
See Table 6-37 for specific synchronization requirements on this register.
<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>6</td><td>VEC</td><td>AltiVec available. e600 core and AltiVec technology specific; optional to the PowerPC ISA.
0 AltiVec technology is disabled.
1 AltiVec technology is enabled.
Note: When a non-stream AltiVec instruction accesses VRs or the VSCR when VEC = 0 an AltiVec unavailable interrupt is generated. This does not occur for data streaming instructions (dst(t), dstst(t), and dss); the VRs and the VSCR are available to data streaming instructions even if VEC = 0. VRSAVE can be accessed even if VEC = 0.</td></tr>
<tr><td>13</td><td>POW</td><td>Power management enable. e600-core-specific and optional to the PowerPC ISA.
0 Power management is disabled.
1 Power management is enabled. The processor can enter a power-saving mode determined by HID0[NAP,SLEEP] when additional conditions are met. See Table 6-7.</td></tr>
<tr><td>29</td><td>PMM</td><td>Performance monitor marked mode. e600-core-specific and optional to the PowerPC ISA. See the “Performance Monitor” chapter of the e600 PowerPC Core Reference Manual.
0 Process is not a marked process.
1 Process is a marked process.</td></tr>
</table></td><td>PEM, 6.1.4.4, e600RM</td></tr>
<tr><td>MSSCR0 [1]</td><td>1014</td><td>Memory subsystem control register. Used to configure and operate many aspects of the core memory subsystem.</td><td>6.1.6.3</td></tr>
<tr><td>MSSSR0 [1]</td><td>1015</td><td>Memory subsystem status register. Used to configure and operate the parity functions in the L2 cache for the e600 core.
See Table 6-37 for specific synchronization requirements on this register.</td><td>6.1.6.4</td></tr>
<tr><td>PIR</td><td>1023</td><td>Processor identification register. Provided for system use. All 32 bits of the PIR can be written to with the mtspr instruction.</td><td>PEM, 6.1.4.3</td></tr>
<tr><td>PMC1–PMC6 [3]</td><td>953, 954 957, 958 945, 946</td><td>Performance monitor counter registers (PMCn). Used to record the number of times a certain event has occurred. UPMCs provide user-level read access to these registers.</td><td>6.1.6.9.8</td></tr>
<tr><td>PTEHI, PTELO</td><td>981, 982</td><td>The PTEHI and PTELO registers are used by the tlbld and tlbli instructions to create a TLB entry. When software table searching is enabled (HID0[STEN] = 1), and a TLB miss interrupt occurs, the bits of the page table entry (PTE) for this access are located by software and saved in the PTE registers.</td><td>6.1.6.7.2 e600RM</td></tr>
<tr><td>PVR</td><td>287</td><td>Processor version register. Read-only register that identifies the version (model) and revision level of the processor.</td><td>PEM 6.1.4.1</td></tr>
</table>

Table 6-1. e600 Core Register Summary (continued)

Name	SPR	Description	Reference/ Section
SDAR, USDAR	—	Sampled data address register. The e600 core does not implement the optional registers (SDAR or the user-level, read-only USDAR register) defined by the PowerPC ISA. Note that in previous processors the SDA and USDA registers could be written to by boot code without causing an interrupt. This is not the case in the e600 core. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program interrupt**.**	6.1.6.9.12
SDR1	25	Sample data register. Specifies the base address of the page table entry group (PTEG) address used in virtual-to-physical address translation. **Implementation Note**—The SDR1 register has been modified (with the SDR1[HTABEXT] and SDR1[HTMEXT] fields) for the e600 core to support the extended 36-bit physical address (when HID0[XAEN] = 1]). See Table 6-37 for specific synchronization requirements on this register.	PEM, 6.1.4.6 e600RM
SIAR [3]	955	Sampled instruction address register. Contains the effective address of an instruction executing at or around the time that the processor signals the performance monitor exception condition. USIAR provides user-level read access to the SIAR.	6.1.6.9.10 e600RM
SPRG0–SPRG3 SPRG4–SPRG7 [1]	272–275 276–279	SPRG*n*. Provided for operating system use. The SPRG4–7 provide additional registers to be used by system software for software table searching.	PEM e600RM
SR0–SR15	—	Segment registers (SR*n*). Note that the e600 core implements separate instruction and data MMUs. It associates architecture-defined SRs with the data MMU. It reflects SRs values in separate, shadow SRs in the instruction MMU. See Table 6-37 for specific synchronization requirements on this register.	PEM
SRR0 SRR1	26 27	Machine status save/restore registers (SRR*n*). Used to save the address of the instruction at which execution continues when **rfi** executes at the end of an interrupt handler routine. SRR1 is used to save machine status on interrupts and to restore machine status when **rfi** executes. **Implementation Note**—When a machine check interrupt occurs, the e600 core sets one or more error bits in SRR1. Refer to the individual interrupts for individual SRR1 bit settings.	PEM, 6.1.4.5 e600RM
SVR	286	System version register. Read-only register provided for future product compatibility.	6.1.4.2
TBL TBU (For writing)	284 285	Time base. A 64-bit structure (two 32-bit registers) that maintains the time of day and operating interval timers. The TB consists of two registers—time base upper (TBU) and time base lower (TBL). The time base registers can be written to only by supervisor-level software. TBL (SPR 284) and TBU (SPR 285) can only be written to and not read from. TBL and TBU can be written to, with the Move to Special Purpose Register (**mtspr**) instruction. **Implementation Note**—Reading from SPR 284 or 285 causes an illegal instruction interrupt. In the e600 core, DEC is decremented and the time base increments at 1/4 of the MPX bus clock frequency.	PEM, 6.1.5.1, 6.3.5.1
TLBMISS [1]	980	The TLBMISS register is automatically loaded when software searching is enabled (HID0[STEN] = 1) and a TLB miss interrupt occurs. Its contents are used by the TLB miss interrupt handlers (the software table search routines) to start the search process.	6.1.6.7.1 e600RM

[1] e600-core-specific register that may not be supported on other cores or processors.

[2] Register is defined by the AltiVec technology.

[3] Defined as optional register in the PowerPC ISA.

The UISA registers are user-level. General-purpose registers (GPRs), floating-point registers (FPRs) and vector registers (VRs) are accessed through instruction operands. Access to registers can be explicit (by using instructions for that purpose such as Move to Special-Purpose Register (**mtspr**) and Move from Special-Purpose Register (**mfspr**) instructions, or implicit as part of the execution of an instruction. Some registers are accessed both explicitly and implicitly.

Implementation Note—The e600 core fully decodes the SPR field of the instruction. If the SPR specified is undefined, an illegal instruction program interrupt occurs.

6.1.4 Supervisor-Level Registers (OEA)

The OEA defines the registers and operating system uses for memory management, configuration, interrupt handling, and other operating system functions as summarized in Table 6-1. The following supervisor-level register defined by the PowerPC ISA contains additional implementation-specific information for the e600 core.

6.1.4.1 Processor Version Register (PVR)

The PVR identifies the version (model) and revision level of the processor. For more information, see "Processor Version Register (PVR)," in Chapter 2, "PowerPC Register Set," of the *Programming Environments Manual*. Table 6-2 shows the PVR settings. The revision level is updated for each silicon revision.

Table 6-2. PVR Settings

Device Name	Version No.	Starting Processor Revision Level
MPC8610	0x8004	*nnnn*

Table 6-3 describes the e600 core PVR bits that are not required by the PowerPC ISA.

Table 6-3. Additional PVR Bits

Bits	Name	Description
0–15	Type	Processor type
16–19	Tech	Processor technology
20–23	Major	Major revision number
24–31	Minor	Minor revision number

6.1.4.2 System Version Register (SVR)

See Section 23.4.1.16, "System Version Register (SVR)," for the specific values of the SVR on the MPC8610.

6.1.4.3 Processor Identification Register (PIR)

For more information, see “Processor Identification Register (PIR),” in Chapter 2, “PowerPC Register Set,” of the *Programming Environments Manual*.

Implementation Note—The e600 core provides write access to the PIR with **mtspr** using SPR 1023.

6.1.4.4 Machine State Register (MSR)

The MSR defines the state of the processor. When an interrupt occurs, MSR bits, as described in Table 6-4 are altered as determined by the interrupts. The MSR can also be modified by the **mtmsr**, **sc**, and **rfi** instructions. It can be read by the **mfmsr** instruction.

The MSR is shown in Figure 6-2.

Access: Read/Write

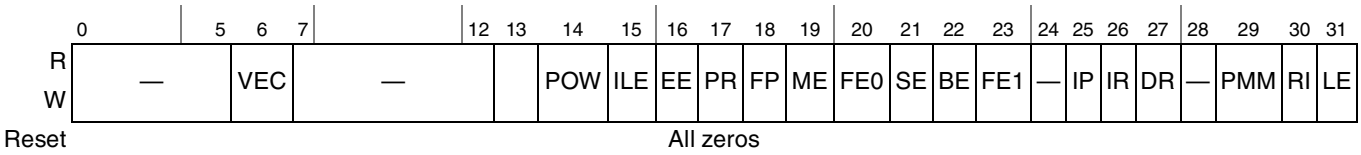


Figure 6-2. Machine State Register (MSR)

The MSR bits are defined in Table 6-4.

Table 6-4. MSR Bit Settings

Bits	Name	Description
0–5	—	Reserved
6	VEC [1, 2]	AltiVec vector unit available 0 The processor prevents dispatch of AltiVec instructions (excluding the data streaming instructions—**dst**, **dstt**, **dstst**, **dststt**, **dss**, and **dssall**). The processor also prevents access to the vector register file (VRF) and the vector status and control register (VSCR). Any attempt to execute an AltiVec instruction that accesses the VRF or VSCR, excluding the data streaming instructions generates the AltiVec unavailable interrupt. The data streaming instructions are not affected by this bit; the VRF and VSCR registers are available to the data streaming instructions even when the MSR[VEC] is cleared. 1 The processor can execute AltiVec instructions and the VRF and VSCR registers are accessible to all AltiVec instructions. Note that the VRSAVE register is not protected by MSR[VEC].
7–12	—	Reserved
13	POW [1, 3]	Power management enable 0 Power management disabled (normal operation mode) 1 Power management enabled (reduced power mode) Power management functions are implementation-dependent. See the “Power and Thermal Management” chapter of the *e600 PowerPC Core Reference Manual*.
14	—	Reserved. Implementation-specific
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.

Table 6-4. MSR Bit Settings (continued)

Bits	Name	Description
16	EE	External interrupt enable 0 The processor delays recognition of interrupts external to the core and decrementer interrupt conditions. 1 The processor is enabled to take an interrupt external to the core or the decrementer interrupt.
17	PR [4]	Privilege level 0 The processor can execute both user- and supervisor-level instructions. 1 The processor can only execute user-level instructions.
18	FP [2]	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 program interrupts.
19	ME	Machine check enable 0 Machine check interrupts are disabled. 1 Machine check interrupts are enabled.
20	FE0 [2]	IEEE Std. 754 floating-point exception mode 0 (see Table 6-5)
21	SE	Single-step trace enable 0 The processor executes instructions normally. 1 The processor generates a single-step trace interrupt upon the successful execution of every instruction except **rfi** and **sc**. Successful execution means that the instruction caused no other interrupt.
22	BE	Branch trace enable 0 The processor executes branch instructions normally. 1 The processor generates a branch type trace interrupt when a branch instruction executes successfully.
23	FE1 [2]	IEEE Std. 754 floating-point exception mode 1 (see Table 6-5)
24	—	Reserved. This bit corresponds to the AL bit of the architecture.
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, *nnnn* 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 [5]	Instruction address translation 0 Instruction address translation is disabled. 1 Instruction address translation is enabled. For more information see the “Memory Management” chapter of the *e600 PowerPC Core Reference Manual*.
27	DR [4]	Data address translation 0 Data address translation is disabled. 1 Data address translation is enabled. For more information see the “Memory Management” chapter of the *e600 PowerPC Core Reference Manual*.
28	—	Reserved

Table 6-4. MSR Bit Settings (continued)

Bits	Name	Description
29	PMM [1]	Performance monitor marked mode 0 Process is not a marked process. 1 Process is a marked process. This bit can be set when statistics need to be gathered on a specific (marked) process. The statistics will only be gathered when the marked process is executing. e600-core-specific; defined as optional by the PowerPC architecture. For more information about the performance monitor marked mode bit, see the “Performance Monitor” chapter of the *e600 PowerPC Core Reference Manual*.
30	RI	Indicates whether system reset or machine check interrupt is recoverable. indicates whether from the perspective of the processor, it is safe to continue (that is, processor state data such as that saved to SRR0 is valid), but it does not guarantee that the interrupted process is recoverable. 0 Interrupt is not recoverable. 1 Interrupt is recoverable.
31	LE [6]	Little-endian mode enable 0 The processor runs in big-endian mode. 1 The processor runs in little-endian mode.

1 Optional to the PowerPC architecture.

2 A context synchronizing instruction must follow an mtmsr instruction.

3 A dssall and sync must precede an mtmsr instruction and then a context synchronizing instruction must follow.

4 A dssall and sync must precede an mtmsr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the MSR[DR] or MSR[PR] bit.

5 A context synchronizing instruction must follow an mtmsr. When changing the MSR[IR] bit the context synchronizing instruction must reside at both the untranslated and the translated address following the mtmsr.

6 A dssall and sync must precede an rfi to guarantee a solid context boundary. Note that if a user is not using AltiVec data streaming instructions, a dssall is not necessary before accessing MSR[LE].

Note that setting MSR[EE] masks not only the architecture-defined interrupt and decrementer interrupts external to the core but also the e600-specific system management, and performance monitor interrupts.

The IEEE Std. 754 floating-point exception mode bits (FE0 and FE1) together define whether floating-point exceptions are handled precisely, imprecisely, or whether they are taken at all. As shown in Table 6-5, if either FE0 or FE1 is set, the e600 core treats interrupts as precise. MSR bits are guaranteed to be written to SRR1 when the first instruction of the interrupt handler is encountered. For further details, see Chapter 2, “Register Set” and Chapter 6, “Interrupts,” of the *Programming Environments Manual*.

Table 6-5. IEEE Std. 754 Floating-Point Exception Mode Bits

FE0	FE1	Mode
0	0	Floating-point exceptions disabled
0	1	Imprecise nonrecoverable. For this setting, the e600 core operates in floating-point precise mode.
1	0	Imprecise recoverable. For this setting, the e600 core operates in floating-point precise mode.
1	1	Floating-point precise mode

6.1.4.5 Machine Status Save/Restore Registers (SRR0, SRR1)

When an interrupt is taken, the processor uses SRR0 and SRR1 to save the contents of the MSR for the current context and to identify where instruction execution should resume after the interrupt is handled.

When an interrupt occurs, the address saved in SRR0 helps determine where instruction processing should resume when the interrupt handler returns control to the interrupted process. Depending on the interrupt, this may be the address in SRR0 or at the next address in the program flow. All instructions in the program flow preceding this one will have completed execution and no subsequent instruction will have begun execution. This may be the address of the instruction that caused the interrupt or the next one (as in the case of a system call or trace interrupt). The SRR0 register is shown in Figure 6-3.

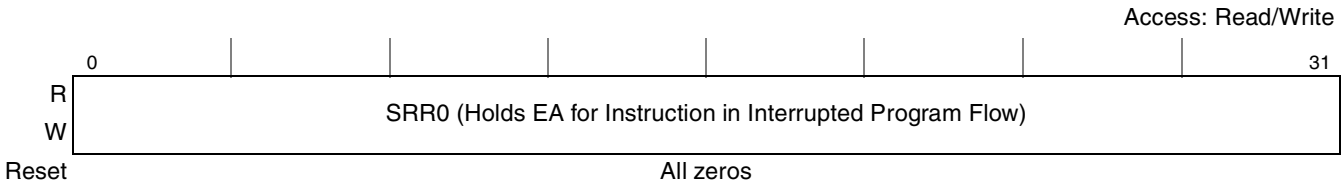


Figure 6-3. Machine Status Save/Restore Register 0 (SRR0)

SRR1 is used to save machine status (selected MSR bits and possibly other status bits) on interrupts and to restore those values when an **rfi** instruction is executed. SRR1 is shown in Figure 6-4.

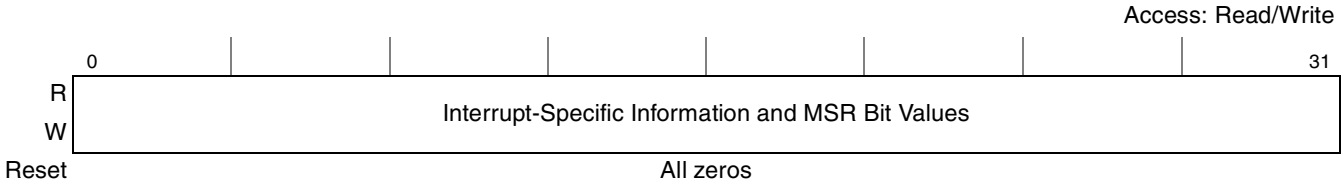


Figure 6-4. Machine Status Save/Restore Register 1 (SRR1)

Typically, when an interrupt occurs, SRR1[0–15] are loaded with interrupt-specific information and MSR[16–31] are placed into the corresponding bit positions of SRR1. For most interrupts, SRR1[0–5] and SRR1[7–15] are cleared, and MSR[6, 16–31] are placed into the corresponding bit positions of SRR1. Table 6-4 provides a summary of the SRR1 bit settings when a machine check interrupt occurs. For a specific interrupt's SRR1 bit settings, see the "Interrupts" chapter of the *e600 PowerPC Core Reference Manual*.

6.1.4.6 SDR1 Register

The SDR1 register specifies the page table entry group (PTEG) address used in virtual-to-physical address translation. See "SDR1," in Chapter 2, "PowerPC Register Set," of the *Programming Environments Manual* for the description with a 32-bit physical address. The SDR1 register has been modified for the e600 core to support the extended 36-bit physical address (when HID0[XAEN] = 1]). See the "Memory Management" chapter of the *e600 PowerPC Core Reference Manual* for details on how SDR1 is modified to support a 36-bit physical address.

Implementation Note—SDR1[HTABEXT] and SDR1[HTMEXT] fields have been added to support extended addressing. The "Memory Management" chapter of the *e600 PowerPC Core Reference Manual*

describes in detail the differences when generating a 36-bit PTEG address. Figure 6-5 shows the format of the modified SDR1.

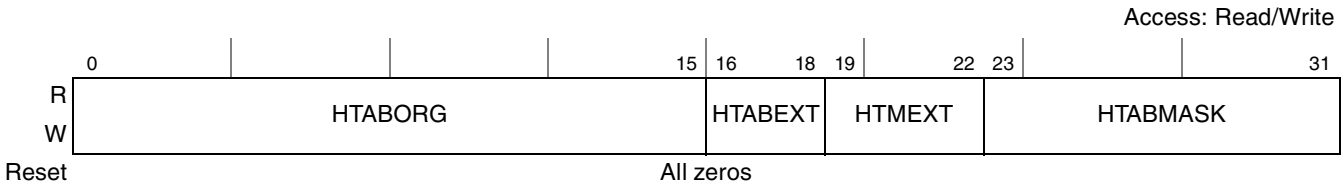


Figure 6-5. SDR1 Register Format—Extended Addressing

Figure 6-6. SDR1 Register Format—Extended Addressing

Bit settings for the SDR1 register are described in Table 6-6.

Table 6-6. SDR1 Register Bit Settings—Extended Addressing

Bits	Name	Description
0–15	HTABORG	Physical base address of page table If HID0[XAEN] = 1, field contains physical address [4–19] If HID0[XAEN] = 0, field contains physical address [0–15]
16–18	HTABEXT	Extension bits for physical base address of page table If HID0[XAEN] = 1, field contains physical address [1–3] If HID0[XAEN] = 0, field is reserved
19–22	HTMEXT	Hash table mask extension bits If HID0[XAEN] = 1, field contains hash table mask [0–3] If HID0[XAEN] = 0, field is reserved
23–31	HTABMASK	Mask for page table address If HID0[XAEN] = 1, field contains hash table mask [4–12] If HID0[XAEN] = 0, field contains hash table mask [0–7]

SDR1 can be accessed with **mtspr** and **mfspr** using SPR 25. For synchronization requirements on accesses to the register see Section 6.3.2.4, “Synchronization.”

6.1.5 User-Level Registers (VEA)

The VEA defines the time base facility (TB), which consists of two 32-bit registers—time base upper (TBU) and time base lower (TBL).

6.1.5.1 Time Base Registers (TBL, TBU)

The time base registers can be written only by supervisor-level instructions but can be read by both user- and supervisor-level software. The time base registers have two different addresses. TBU and TBL can be read from the TBR 268 and 269, respectively, with the move from time base register (**mftb**) instruction. TBU and TBL can be written to TBR 284 and 285, respectively, with the move to special purpose register (**mtspr**) instruction. Reading from SPR 284 or 285 causes an illegal instruction interrupt. For more information, see “PowerPC VEA Register Set—Time Base,” in Chapter 2, “PowerPC Register Set,” of the

Programming Environments Manual. Note that DEC is decremented and the time base increments at 1/4 of the MPX bus clock frequency.

6.1.6 e600-Core-Specific Register Descriptions

The PowerPC ISA allows for implementation-specific SPRs. This section describes registers that are defined for the e600 core but are not included in the PowerPC ISA. Note that in the e600 core, these registers are all supervisor-level registers. All the registers described in the *AltiVec Technology Programming Environments Manual* are implemented in e600 core. See Chapter 2, “AltiVec Register Set,” in the *AltiVec Technology Programming Environments Manual* for details about these registers.

Note that while it is not guaranteed that the implementation of e600-core-specific registers is consistent among processors that implement the PowerPC ISA, other processors can implement similar or identical registers.

The registers in the following subsections are presented in the order of the chapters in this book. First, the processor control registers are described followed by the cache control registers. Next, the implementation-specific registers for interrupt processing and memory management are presented, followed by the thermal management register. Finally, the performance monitor registers are presented.

6.1.6.1 Hardware Implementation-Dependent Register 0 (HID0)

The hardware implementation-dependent register 0 (HID0) controls the state of several functions within the e600 core. The HID0 register is shown in Figure 6-7.

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	—					TBEN	—	STEN	HIGH_BAT_EN	NAP	SLEEP	DPM	—	BHTCLR	XAEN	NHR
Reset	1	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	ICE	DCE	ILOCK	DLOCK	ICFI	DCFI	SPD	XBSEN	SGE	—	BTIC	LRSTK	FOLD	BHT	NOPDST	NOPTI
Reset	All zeros															

Figure 6-7. Hardware Implementation-Dependent Register 0 (HID0)

The HID0 bits are described in Table 6-7.

Table 6-7. HID0 Field Descriptions

Bits	Name	Description
0–4	—	Reserved. Read as 0b1000_0.
5	TBEN [1]	Time base enable. Note that this bit must be set and the *tben* signal must be asserted to enable the time base and decrementer.
6	—	Reserved.

Table 6-7. HID0 Field Descriptions (continued)

Bits	Name	Description
7	STEN[2]	Software table search enable. When a TLB miss occurs, the e600 core takes one of three TLB miss interrupts so that software can search the page tables for the desired PTE. See the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual* for details on the e600 core facilities for software table searching. 0 Hardware table search enabled 1 Software tables search enabled
8	HIGH_BAT_EN	Additional BATs enabled for the e600 core 0 Additional 4 IBATs (4–7) and 4 DBATs (4–7) disabled 1 Additional 4 IBATs (4–7) and 4 DBATs (4–7) enabled The additional BATs provide for more mapping of memory with the block address translation method.
9	NAP [1]	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 the time base remain active. Note that if both NAP and SLEEP are set, the e600 core ignores the SLEEP bit.
10	SLEEP [1]	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. $\overline{qreq}$ is asserted to indicate that the processor is ready to enter sleep mode. If the system logic determines that the core can enter sleep mode, the quiesce acknowledge signal, $\overline{qack}$, is asserted back to the core. When the $\overline{qack}$ signal assertion is detected, the core enters sleep mode after several core clocks. At this point, the system logic can turn off the PLL by first configuring *pll_cfg*[0:5] to PLL bypass mode, and then disabling *sysclk*.
11	DPM [1]	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 hardware external to the core.
12	—	Reserved. For test use; software should not set this bit.
13	BHTCLR [3]	Clear branch history table 0 The e600 core clears this bit one cycle after it is set. 1 Setting BHTCLR bit initializes all entries in BHT to weakly, not taken whether or not the BHT is enabled by HID0[BHT]. However, for correct results, the BHT should be disabled (HID0[BHT] = 0) before setting BHTCLR. Setting BHTCLR causes the branch unit to be busy for 64 cycles while the initialization process is completed.
14	XAEN [4]	Extended addressing enabled 0 Extended addressing is disabled; the 4 most significant bits of the 36-bit physical address are cleared and a 32-bit physical address is used. 1 Extended addressing is enabled; the 32-bit effective address is translated to a 36-bit physical address. If HID0[XAEN] is changed (cleared or set), the BATs and TLBs must be invalidated first.
15	NHR [1]	Not hard reset (software-use only). Helps software distinguish a hard reset from a soft reset. 0 A hard reset occurred if software had previously set this bit. 1 A hard reset has not occurred. If software sets this bit after a hard reset, when a reset occurs and this bit remains set, software knows it was a soft reset. The e600 core never writes this bit unless executing an **mtspr**(HID0).

Table 6-7. HID0 Field Descriptions (continued)

Bits	Name	Description
16	ICE [5]	Instruction cache enable 0 The instruction cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIMG = x1xx). Potential cache accesses from the MPX bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the MPX bus as burst transactions. For those transactions, $\overline{ci}$ is asserted regardless of address translation. ICE is zero at power-up. 1 The instruction cache is enabled. Note that HID0[ICFI] must be set at the same time that this bit is set.
17	DCE [2]	Data cache enable 0 The data cache is neither accessed nor updated. All pages are accessed as if they were marked cache-inhibited (WIMG = x1xx). Potential cache accesses from the MPX bus (snoop and cache operations) are ignored. In the disabled state for the L1 caches, the cache tag state bits are ignored and all accesses are propagated to the L2 cache or MPX bus as cache-inhibited. For those transactions, $\overline{ci}$ is asserted regardless of address translation. DCE is zero at power-up. 1 The data cache is enabled. Note that HID0[DCFI] must be set at the same time that this bit is set.
18	ILOCK [6]	Instruction cache lock 0 Normal operation 1 All of the ways of the instruction cache are locked. A locked cache supplies data normally on a read hit. On a miss, the access is treated the same as if the instruction cache was disabled.Thus, the MPX bus request is a 32-byte burst read, but the cache is not loaded with data. The data is reloaded into the L2, unless the L2CR[L2DO] bit is set. Note that setting this bit has the same effect as setting ICTRL[ICWL] to all ones. However, when this bit is set, ICTRL[ICWL] is ignored. the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* gives further details.
19	DLOCK [2]	Data cache lock 0 Normal operation 1 All the ways of the data cache are locked. A locked cache supplies data normally on a read hit but is treated as a cache-inhibited transaction on a miss. On a miss, a load transaction still reads a full cache line from the L2 or MPX bus but does not reload that line into the L1. Any store miss is treated like a write-through store and the transaction occurs on the MPX bus with the $\overline{wt}$ signal asserted. A snoop hit to a locked L1 data cache operates as if the cache were not locked. A cache block invalidated by a snoop remains invalid until the cache is unlocked. Note that setting this bit has the same effect as setting LDSTCR[DCWL] to all ones. However, when this bit is set, LDSTCR[DCWL] is ignored. Refer to the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for further details. To prevent locking during a cache access, a **sync** instruction must precede the setting of DLOCK and a **sync** must follow.
20	ICFI [5]	Instruction cache flash invalidate 0 The instruction cache is not invalidated. The bit is cleared when the invalidation operation begins (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 sets the PLRU bits to point to way L0 of each set. When the L1 flash invalidate bits are set through an **mtspr** operation, the hardware automatically clears these bits in the next cycle (provided that the corresponding cache enable bits are set in HID0). Note that in the MPC603e processors, the proper use of the ICFI and DCFI bits was to set and clear them in two consecutive **mtspr** operations. Software that already has this sequence of operations does not need to be changed to run on the e600 core.

Table 6-7. HID0 Field Descriptions (continued)

Bits	Name	Description
21	DCFI [2]	Data cache flash invalidate 0 The data cache is not invalidated. The bit is cleared when the invalidation operation begins (the next cycle after the write operation to the register). 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. MPX 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. When the L1 flash invalidate bits are set through an **mtspr** operation, the hardware automatically clears these bits in the next cycle. Note that setting DCFI invalidates the data cache regardless of whether it is enabled. Note that in the MPC603e processors, the proper use of the ICFI and DCFI bits was to set them and clear them in two consecutive **mtspr** operations. Software that already has this sequence of operations does not need to be changed to run on the e600 core.
22	SPD [1]	Speculative data cache and instruction cache access disable 0 Speculative bus accesses to nonguarded space (G = 0) from both the instruction and data caches is enabled. 1 Speculative bus accesses to nonguarded space in both caches is disabled. Thus, setting this bit prevents L1 data cache misses from going to the core memory subsystem until the instruction that caused the miss is next to complete. The HID0[SPD] bit also prevents instruction cache misses from going to the core memory subsystem until there are no unresolved branches. For more information on this bit and its effect on re-ordering of loads and stores, see the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual.*
23	XBSEN	Extended BAT block size enable. 0 Disables IBAT*n*U[XBL] and DBAT*n*U[XBL] bits and clears these bits to zero. 1 Enables IBAT*n*U[XBL] and DBAT*n*U[XBL] bits. BATnU[15–18] become the 4 MSBs of the extended 15-bit BL field (BATnU[15–29]). This allows for extended BAT block sizes of 512 Mbytes, 1 Gbyte, 2 Gbytes, and 4 Gbytes. If HID0[XBSEN] is set at startup and then cleared after startup, the XBL bits will not clear but stay the same as they were set at startup. HID0[XBSEN] should be set once at startup and once set should not be cleared. When HID0[XBSEN] is set at startup, and then HID0[XBSEN] is cleared, the IBAT*n*U[XBL] and DBAT*n*U[XBL] bits are not cleared but stay the same as what was set at startup. If backwards compatibility with previous processors is a concern, then HID0[XBSEN] should stay cleared so that the XBL bits are treated as zeros. This allows the BAT translation to have a maximum block length of 256 Mbytes.
24	SGE [7]	Store gathering enable 0 Store gathering is disabled. 1 Integer store gathering is performed as described in the “L1 and L2 Cache Operation” and “Instruction Timing” chapters of the *e600 PowerPC Core Reference Manual*.
25	—	Reserved. Defined as DCFA on some earlier processors.
26	BTIC [1]	Branch target instruction cache enable. Used to enable use of the 128-entry branch instruction cache. 0 The BTIC contents are invalidated and the BTIC behaves as if it were empty. New entries cannot be added until the BTIC is enabled. 1 The BTIC is enabled and new entries can be added. The BTIC is flushed by context synchronization, which is required after a move to HID0. Thus, if the synchronization rules are followed, modifying this BTIC bit implicitly flushes the BTIC. See the “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual* for further details.

Table 6-7. HID0 Field Descriptions (continued)

Bits	Name	Description
27	LRSTK [1]	Link register stack enable 0 Link register prediction is disabled. 1 Allows **bclr** and **bclrl** instructions to predict the branch target address using the link register stack which can accelerate returns from subroutines. See the “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual* for further details.
28	FOLD [1]	Branch folding enable 0 Branch folding is disabled. All branches are dispatched to the completion queue. 1 Branch folding is enabled, allowing branches to be folded out of the instruction prefetch stream before dispatch. The e600 core attempts to fold branches that do not modify the link and or count register. Note that a branch that is taken or predicted as taken is folded regardless of where it is in the IQ; however, a branch that is predicted as not taken must be dispatched (cannot be fall-through folded) if it is in one of the dispatch entries (IQ0–IQ2) the cycle after it is decoded. See the “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual* for further details.
29	BHT [1]	Branch history table enable 0 BHT disabled. The e600 core uses static branch prediction as defined by the PowerPC architecture (UISA) for those branch instructions the BHT would have otherwise used to predict (that is, those that use the CR or CTR mechanism to determine direction). For more information on static branch prediction, see “Conditional Branch Control,” in Chapter 4 of the *Programming Environments Manual*. 1 Allows the use of dynamic prediction in the 2048-entry branch history table (BHT). The BHT is disabled at power-on reset. All entries are set to weakly, not-taken.
30	NOPDST [2]	No-op **dst**, **dstt**, **dstst**, and **dststt** instructions 0 The **dst**, **dstt**, **dstst**, and **dststt** instructions are enabled. 1 The **dst**, **dstt**, **dstst**, and **dststt** instructions are no-oped globally, and all previously executed **dst** streams are cancelled.
31	NOPTI [7]	No-op the data cache touch instructions 0 The **dcbt** and **dcbtst** instructions are enabled. 1 The **dcbt** and **dcbtst** instructions are no-oped globally.

[1] A context synchronizing instruction must follow the mtspr.

[2] A dssall and sync must precede an mtspr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the HID0[DCE] or HID0[DCFI] bit.

[3] A context synchronizing instruction must precede an mtspr and a branch instruction should follow. The branch instruction may be either conditional or unconditional. It ensures that all subsequent branch instructions see the newly initialized BHT values. For correct results, the BHT should be disabled (HID0[BHT] = 0) before setting BHTCLR.

[4] A dssall and sync must precede an mtspr and then a sync and a context-synchronizing instruction must follow. Alteration of HID0[XAEN] must be done with caches and translation disabled. The caches and TLBs must be flushed before they are re-enabled after the XAEN bit is altered. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the HID0[XAEN] bit.

[5] A context synchronizing instruction must immediately follow an mtspr. An mtspr instruction for HID0 should not modify either of these bits at the same time it modifies another bit that requires additional synchronization.

[6] A context synchronizing instruction must precede and follow an mtspr.

[7] An mtspr must follow a sync and a context synchronizing instruction.

HID0 can be accessed with **mtspr** and **mfspr** using SPR 1008. All **mtspr** instructions should be followed by a context synchronization instruction such as **isync**; for specific details see Section 6.3.2.4, “Synchronization.”

6.1.6.2 Hardware Implementation-Dependent Register 1 (HID1)

The hardware implementation-dependent register 1 (HID1) reflects the state of the *pll_cfg*[0:5] signals and controls other functions. At reset, PC bits 0–5 reflect the values on the *pll_cfg*[0:5] pins. The HID1 bits are shown in Figure 6-8.

Access: Read/Write

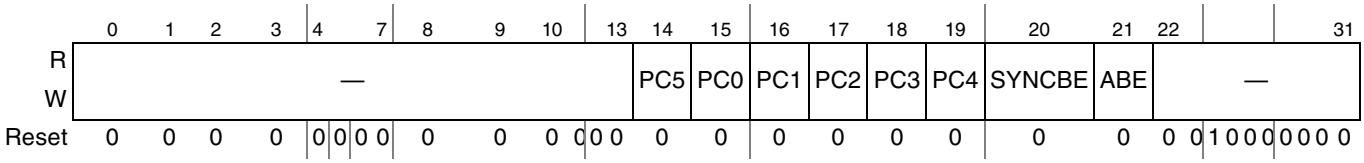


Figure 6-8. Hardware Implementation-Dependent Register 1 (HID1)

Figure 6-9. Hardware Implementation-Dependent Register 1 (HID1)

The HID1 bits are described in Table 6-8.

Table 6-8. HID1 Field Descriptions

Bits[1]	Name	Description
0	EMCP	Machine check signal enable 0 Machine check is disabled. 1 Machine check input signal ($\overline{mcp}$) is enabled to cause machine check errors or checkstops.
1–13	—	Reserved
14	PC5	PLL configuration bit 5 (read-only). Reflects the state of the PLL multiplier.
15	PC0	PLL configuration bit 0 (read-only). Reflects the state of the PLL multiplier.
16	PC1	PLL configuration bit 1 (read-only). Reflects the state of the PLL multiplier.
17	PC2	PLL configuration bit 2 (read-only). Reflects the state of the PLL multiplier.
18	PC3	PLL configuration bit 3 (read-only). Reflects the state of the PLL multiplier.
19	PC4	PLL configuration bit 4 (read-only). Reflects the state of the PLL multiplier.
20	SYNCBE	Address broadcast enable for **sync**, **eieio** 0 Address broadcasting of **sync**, and **eieio** is disabled. 1 Address broadcasting of **sync**, and **eieio** is enabled. Note this bit must be set in MP systems and systems that reorder stores.
21	ABE	Address broadcast enable for **dcbf**, **dcbst**, **dcbi**, **icbi**, **tlbie**, and **tlbsync**. 0 Address broadcasting of **dcbf**, **dcbst**, **dcbi**, **icbi**, **tlbie**, and **tlbsync** is disabled. Note that when HID1[ABE] is cleared this does not exclude all cache operations from the MPX bus—just **icbi**, **tlbie**, and **tlbsync**. 1 Address broadcasting for cache control operations (**dcbf**, **dcbst**, **dcbi**, **icbi**) and TLB control operations (**tlbie** and **tlbsync**) is enabled. Note that whether the broadcast occurs depends on the setting of the M bit of WIMG and whether the access causes a hit to modified memory. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for more information on broadcast operations. The ABE bit must be set for MP systems.
22–31	—	Reserved. Read as 0b00_1000_0000.

[1] A sync and context synchronizing instruction must follow an mtspr.

HID1 can be accessed with **mtspr** and **mfspr** using SPR 1009. All **mtspr** instructions should be followed by a **sync** and context synchronization instruction; for specific details, see Section 6.3.2.4, "Synchronization."

6.1.6.3 Memory Subsystem Control Register (MSSCR0)

The memory subsystem control register (MSSCR0), shown in Figure 6-10, is used to configure and operate the memory subsystem for the e600 core. It is accessed as SPR 1014. The MSSCR0 is initialized to all zeros except for the read-only bits.

Because MSSCR0 alters how the e600 responds to snoop requests, it is important that changes to the values of the fields in MSSCR0 are handled correctly.

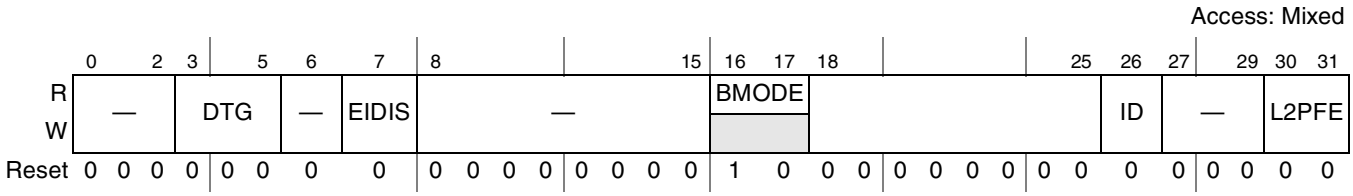


Figure 6-10. Memory Subsystem Control Register (MSSCR0)

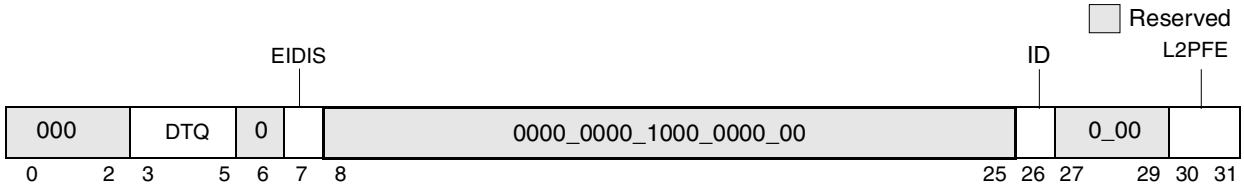


Table 6-9 describes the MSSCR0 fields.

Table 6-9. MSSCR0 Field Descriptions

Bits	Name	Function
0–2	—	Reserved
3–5	DTQ	DTQ size. Determines the maximum number of outstanding data bus transactions that the e600 core can support. The DTQ bit values are as follows: 000 8 entries 001 16 entries 010 2 entries 011 3 entries 100 4 entries 101 5 entries 110 6 entries 111 7 entries The value of this field must match what the integrated device supports.
6	—	Reserved

Table 6-9. MSSCR0 Field Descriptions (continued)

Bits	Name	Function
7	EIDIS	Disable external intervention 0 Interventions external to the core occur. 1 The e600 core performs external pushes instead of external interventions. External interventions are disabled.
8–15	—	Reserved
16–17	BMODE	Bus mode (read-only). Indicates whether the core interface uses the 60x or MPX bus protocol as described in the “Core Interface” chapter of the *e600 PowerPC Core Reference Manual*. 00 Reserved 01 Reserved 10 MPX bus mode 11 Reserved
18–25	—	Reserved. Normally cleared. Used in debug. Writing non-zero values may cause boundedly undefined results.
26	ID	Processor identification. Sets the processor ID to either processor 0 or 1. Determined by the integration logic. Software can then find processor 0 and use it to re-identify the other processors by writing unique values to the PIR of the other CPUs.
27–29	—	Reserved. Read as zeros.
30–31	L2PFE	L2 prefetching enabled. The following values determine the number of L2 prefetch engines enabled as follows: 00 L2 prefetching disabled, no prefetch engines 01 One prefetch engine enabled 10 Two prefetch engines enabled 11 Three prefetch engines enabled These bits enable alternate sector prefetching in the 2-sectored L2 cache; up to 3 outstanding prefetch engines may be active. See the integrated device reference manual for specific recommendations.

6.1.6.4 Memory Subsystem Status Register (MSSSR0)

The memory subsystem status register (MSSSR0), shown in Figure 6-11, is used to report parity in the L2 cache and MSS enabled error status. It is accessed as SPR 1015. The MSSSR0 is initialized to all 0s except for the read-only bits.

Note that tag and data parity and data ECC errors are reported in the error detect (L2ERRDET) register whether or not error reporting is enabled. The corresponding bit in MSSSR0 is set only if error reporting is enabled.

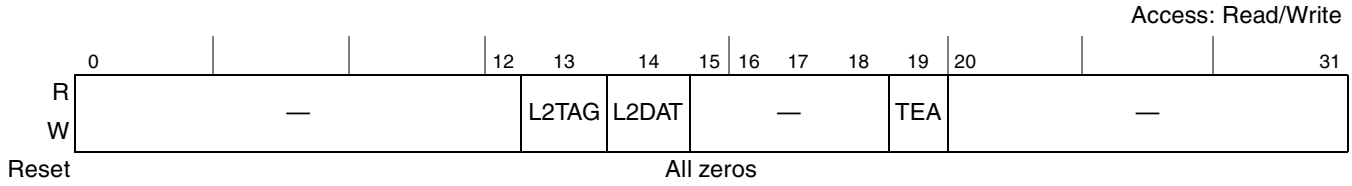


Figure 6-11. MSS Status Register (MSSSR0)

Table 6-10 describes MSSSR0 fields.

Table 6-10. MSSSR0 Field Descriptions

Bits	Name	Description
0–12	—	Reserved. Normally cleared. Used in debug. Writing nonzero values may cause boundedly undefined results.
13	L2TAG	L2 tag parity error 0 L2 tag parity error not detected. 1 L2 tag parity error detected.
14	L2DAT	L2 data parity error 0 L2 data parity error not detected. 1 L2 data parity error detected.
15–18	—	Reserved. Normally cleared. Used in debug. Writing nonzero values may cause boundedly undefined results.
19	TEA	Bus transfer error acknowledge 0 $\overline{tea}$ not detected as asserted. 1 $\overline{tea}$ detected as asserted.
20–31	—	Reserved

6.1.6.5 Instruction and Data Cache Registers

Several registers are used for configuring and controlling the various L1 and L2 caches. Along with the cache registers (L2CR, ICTRL, and LDSTCR), HID0 is used in configuring the caches. Details of how the various cache registers are used is discussed below. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for further details on configuring the cache.

6.1.6.5.1 L2 Cache Control Register (L2CR)

The L2 cache control register (L2CR), shown in Figure 6-12, is a supervisor-level, implementation-specific SPR used to configure and operate the L2 cache. It is cleared by a hard reset or power-on reset.

With the addition of ECC support, tag parity is controlled separately through the TPARDIS bit in the L2ERRDIS register. Data parity can only be enabled through L2CR[L2PE] if ECC is disabled in the L2ERRDIS register.

Access: Read/Write

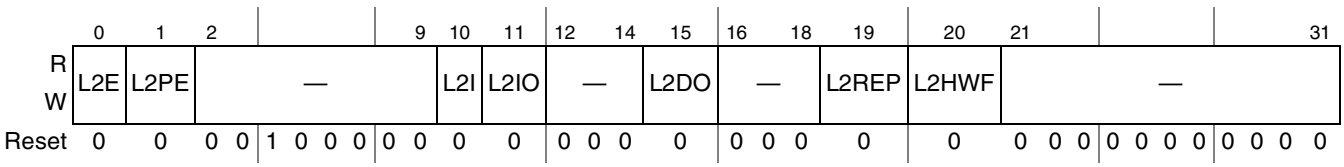


Figure 6-12. L2 Control Register (L2CR)

The L2 cache interface is described in the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*. The bits of the L2CR are described in Table 6-11.

Table 6-11. L2CR Field Descriptions

Bits	Name	Description
0	L2E	L2 enable. Used to enable the L2 cache. 0 The L2 cache is disabled and is not accessed for reads, snoops, or writes. 1 The L2 cache is enabled.
1	L2PE	L2 data parity checking enable 0 L2 data parity checking disabled 1 L2 data parity checking enabled if L2ERRDIS[MBECCDIS] = 1 and L2ERRDIS[SBECCDIS] = 1. If ECC is enabled (L2ERRDIS[MBECCDIS] = 0 or L2ERRDIS[SBECCDIS] = 0), setting L2PE has no effect; ECC checking will still be performed. Note: The L2ERRDIS register includes bits to enable/disable tag parity checking. Data parity can only be enabled with L2CR[L2PE] if ECC is disabled in the L2ERRDIS register. By default, tag parity and data ECC checking are enabled on the e600 core.
2–3	—	Reserved, will read as 0b01.
4–9	—	Reserved
10	L2I	L2 global invalidate 0 Do not perform global invalidate operation on the L2 cache. 1 L2 cache invalidate globally Invalidates the L2 cache globally by clearing the L2 status bits. This bit must not be set while the L2 cache is enabled. Note that L2I is automatically cleared when the global invalidate operation completes.
11	L2IO	L2 instruction-only. Causes the L2 cache to allocate lines for instruction cache transactions only. 0 The L2 cache allocates entries for data accesses that miss. 1 The L2 cache does not allocate entries for data accesses that miss in the L2. Data accesses that hit, instruction accesses, and system accesses are unaffected. If L2DO and L2IO are both set, no new lines are allocated in the L2 cache, effectively locking the entire cache.
12–14	—	Reserved
15	L2DO	L2 data only. L2 cache lines are allocated for data cache transactions only. 0 The L2 cache allocates entries for instruction accesses that miss. 1 The L2 cache does not allocate entries for instruction accesses that miss in the L2. Instruction accesses that hit in the L2, data accesses, and system accesses are unaffected. If both L2DO and L2IO are set, no new lines are allocated in the L2 cache, effectively locking the entire cache.
16–18	—	Reserved
19	L2REP	L2 replacement algorithm selection on a miss 0 Pseudo-random replacement algorithm is used (default) 1 3-bit counter replacement algorithm is used See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for more information.
20	L2HWF	L2 hardware flush 0 The L2 cache is not being flushed. 1 A 0->1 transition on this bit triggers a global flush of the entire L2 cache. All modified lines will be cast out to main memory. The cache must be locked by setting L2CR[L2DO] = 1 and L2CR[L2IO] = 1 before setting this bit. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for more information.

Table 6-11. L2CR Field Descriptions (continued)

Bits	Name	Description
21–23	—	Reserved
24–27	—	Reserved. Writing these bits will result in undefined L2 cache behavior.
28–31	—	Reserved

The L2CR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1017.

6.1.6.5.2 L2 Error Injection Mask High Register (L2ERRINJHI)

The L2 error injection mask high register (L2ERRINJHI), shown in Figure 6-13, is a supervisor-level SPR used for error injection of the high word of the data path.

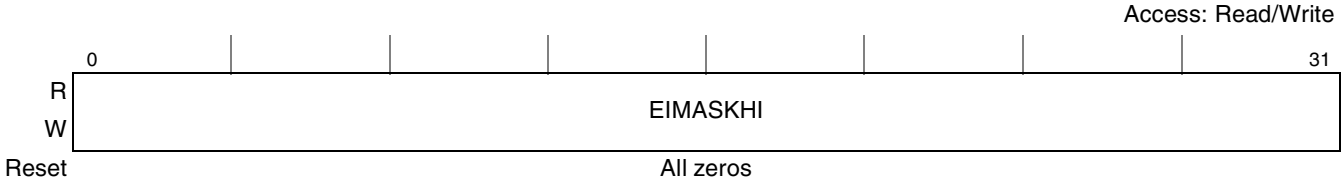


Figure 6-13. L2 Error Injection Mask High Register (L2ERRINJHI)

Table 6-12 describes L2ERRINJHI[EIMASKHI].

Table 6-12. L2ERRINJHI Field Description

Bits	Name	Description
0–31	EIMASKHI	Error injection mask for the high word of the data path. A set bit corresponding to a data path bit causes that bit on the data path to be inverted on cache writes if date array error injection is enabled by setting L2ERRINJCTL[DERRIEN] = 1.

6.1.6.5.3 L2 Error Injection Mask High Register (L2ERRINJLO)

The L2 error injection mask low register (L2ERRINJLO), shown in Figure 6-14, is a supervisor-level SPR used for error injection of the low word of the data path.

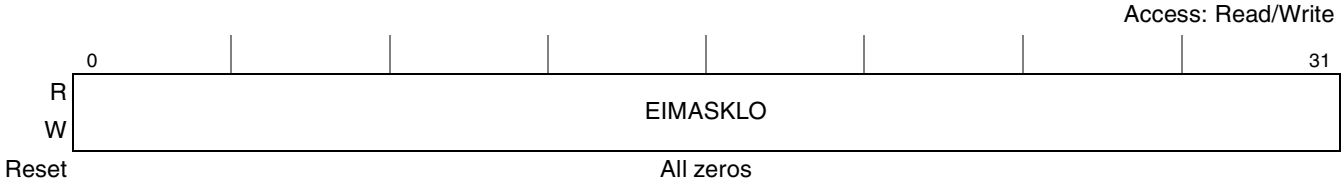


Figure 6-14. L2 Error Injection Mask Low Register (L2ERRINJLO)

Table 6-13 describes L2ERRINJLO[EIMASKLO].

Table 6-13. L2ERRINJLO Field Description

Bits	Name	Description
0–31	EIMASKLO	Error injection mask for the low word of the data path. A set bit corresponding to a data path bit causes that bit on the data path to be inverted on cache writes if data array error injection is enabled by setting L2ERRINJCTL[DERRIEN] = 1.

6.1.6.5.4 L2 Error Injection Mask Control Register (L2ERRINJCTL)

The L2 error injection mask control register (L2ERRINJCTL), shown in Figure 6-15, is a supervisor-level SPR used to configure error injection.

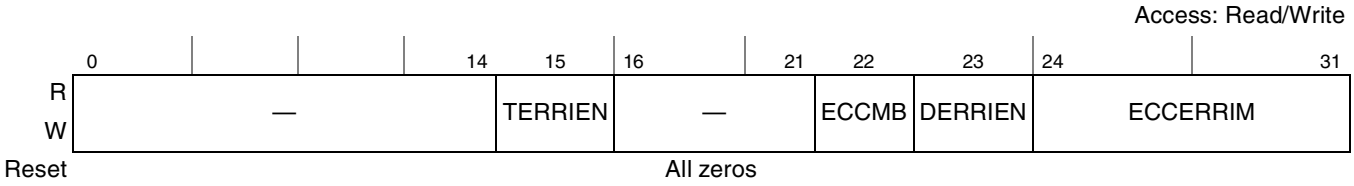


Figure 6-15. L2 Error Injection Mask Control Register (L2ERRINJCTL)

Table 6-14 describes L2ERRINJCTL fields.

Table 6-14. L2ERRINJCTL Field Descriptions

Bits	Name	Description
0–14	—	Reserved
15	TERRIEN	L2 tag array error injection enable 0 No tag errors are injected. 1 All subsequent entries written to the L2 tag array have the parity bit inverted.
16–21	—	Reserved
22	ECCMB	ECC mirror byte enable 0 ECC byte mirroring is disabled. 1 The most significant data path byte is mirrored onto the ECC byte if DERRIEN = 1.
23	DERRIEN	L2 data array error injection enable 0 No data errors are injected. 1 All subsequent entries written to the L2 data array have data or ECC bits inverted as specified in the data error injection masks and ECC error injection mask and/or data path byte mirrored onto ECC as specified by the ECC mirror byte enable bit, ECCMB. Note: if both ECC mirror byte and data error injection are enabled, ECC mask error injection is performed on the mirrored ECC.
24–31	ECCERRIM	Error injection mask for the ECC bits. A set bit corresponding to an ECC bit causes that bit to be inverted on cache writes if DERRIEN = 1.

6.1.6.5.5 L2 Error Capture Data High Register (L2CAPTDATAHI)

The L2 error capture data high register (L2CAPTDATAHI), shown in Figure 6-17, holds the high word of the L2 data that contains the detected error.

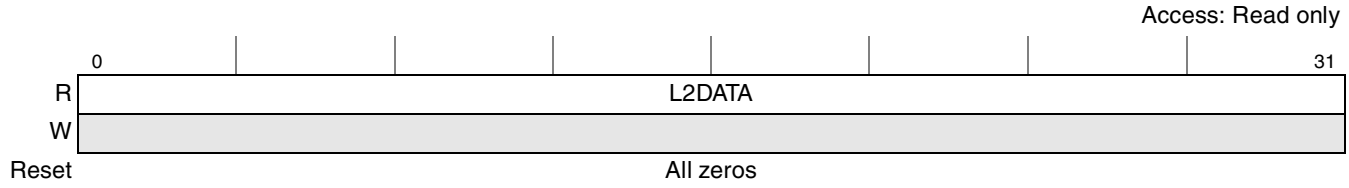


Figure 6-16. L2 Error Capture Data High Register (L2CAPTDATAHI)

Table 6-15 describes L2CAPTDATAHI[L2DATA].

Table 6-15. L2CAPTDATAHI Field Description

Bits	Name	Description
0–31	L2DATA	L2 data high word (read only)

6.1.6.5.6 L2 Error Capture Data Low Register (L2CAPTDATALO)

The L2 error capture data low register (L2CAPTDATALO), shown in Figure 6-17, holds the low word of the L2 data that contains the detected error.

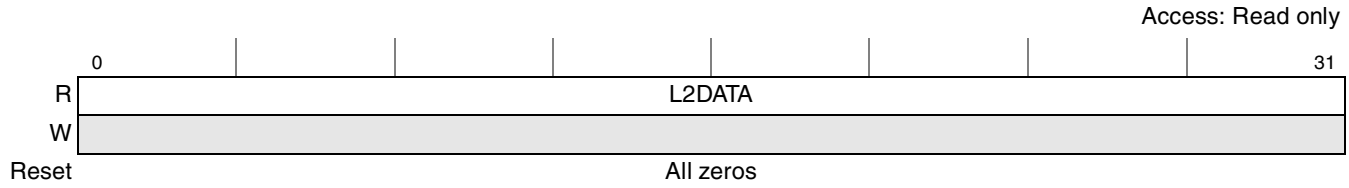


Figure 6-17. L2 Error Capture Data Low Register (L2CAPTDATALO)

Table 6-16 describes L2CAPTDATALO[L2DATA].

Table 6-16. L2CAPTDATALO Field Description

Bits	Name	Description
0–31	L2DATA	L2 data low word (read only)

6.1.6.5.7 L2 Error Syndrome Register (L2CAPTECC)

The L2 error syndrome register (L2CAPTECC), shown in Figure 6-18, is a supervisor-level SPR that contains the ECC syndrome and datapath ECC of the failing double word.

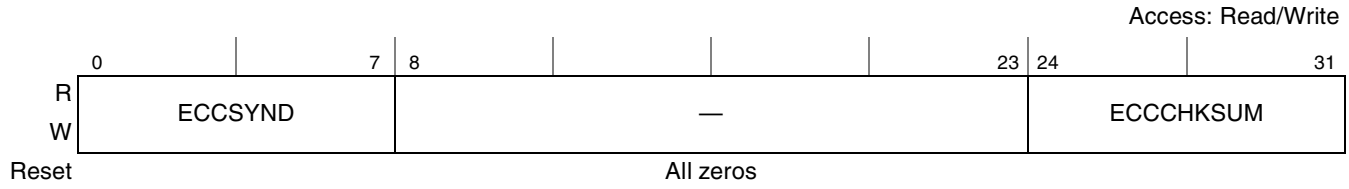


Figure 6-18. L2 Error Syndrome Register (L2CAPTECC)

Table 6-17 describes L2CAPTECC fields.

Table 6-17. L2CAPTECC Field Descriptions

Bits	Name	Description
0–7	ECCSYND	The calculated ECC syndrome of the failing double word (read only)
8–23	—	Reserved
24–31	ECCCHKSUM	The datapath ECC of the failing double word (read only)

6.1.6.5.8 L2 Error Detect Register (L2ERRDET)

The L2 error detect register (L2ERRDET), shown in Figure 6-19, is a supervisor-level SPR that shows the errors detected.

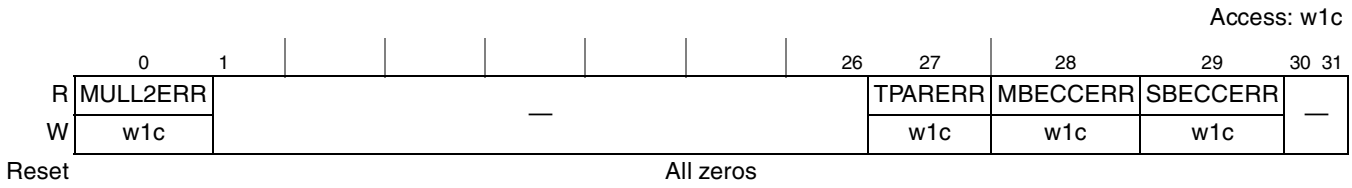


Figure 6-19. L2 Error Detect Register (L2ERRDET)

Table 6-18 describes L2ERRDET fields.

Table 6-18. L2ERRDET Field Descriptions

Bits	Name	Description
0	MULL2ERR	Multiple L2 errors (Bit reset, write-1-to-clear) 0 Multiple L2 errors of the same type were not detected 1 Multiple L2 errors of the same type were detected Note that setting this bit to 1 clears it to a value of 0.
1–26	—	Reserved
27	TPARERR	Tag parity error (Bit reset, write-1-to-clear) 0 Tag parity error was not detected 1 Tag parity error was detected Note that setting this bit to 1 clears it to a value of 0.
28	MBECCERR	Multiple-bit ECC error (Bit reset, write-1-to-clear) 0 Multiple-bit ECC errors were not detected 1 Multiple-bit ECC errors were detected Note that setting this bit to 1 clears it to a value of 0.
29	SBECCERR	Single-bit ECC error (Bit reset, write-1-to-clear) 0 Single-bit ECC error was not detected 1 Single-bit ECC error was detected Note that setting this bit to 1 clears it to a value of 0.
30–31	—	Reserved

6.1.6.5.9 L2 Error Disable Register (L2ERRDIS)

The L2 error disable register (L2ERRDIS), shown in Figure 6-20, is a supervisor-level SPR that disables and enables error detection. Note that the L2 cache must be disabled and flushed before enabling or disabling ECC to ensure that no errors occur. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* and Table 6-37 for the synchronization requirements required to enable or disable ECC.

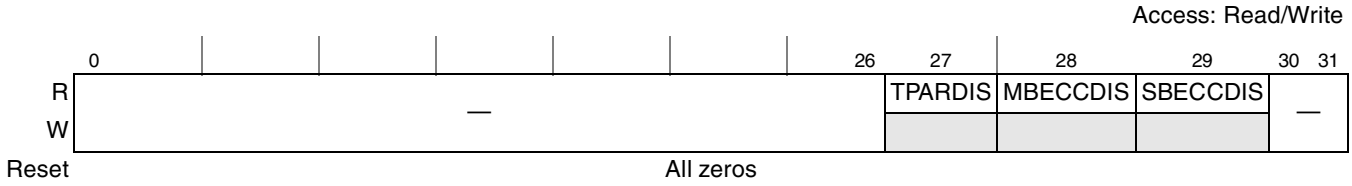


Figure 6-20. L2 Error Disable Register (L2ERRDIS)

Table 6-19 describes L2ERRDIS fields.

Table 6-19. L2ERRDIS Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27	TPARDIS	Tag parity error disable 0 Tag parity error detection enabled 1 Tag parity error detection disabled
28	MBECCDIS	Multiple-bit ECC error disable 0 Multiple-bit ECC error detection enabled 1 Multiple-bit ECC error detection disabled
29	SBECCDIS	Single-bit ECC error disable 0 Single-bit ECC error detection enabled 1 Single-bit ECC error detection disabled
30–31	—	Reserved

6.1.6.5.10 L2 Error Interrupt Enable Register (L2ERRINTEN)

The L2 error interrupt enable register (L2ERRINTEN), shown in Figure 6-21, is a supervisor-level SPR used to enable L2 error interrupts. When any of these error conditions exist and the corresponding bit in the L2ERRINTEN register is enabled, a machine check interrupt is generated.

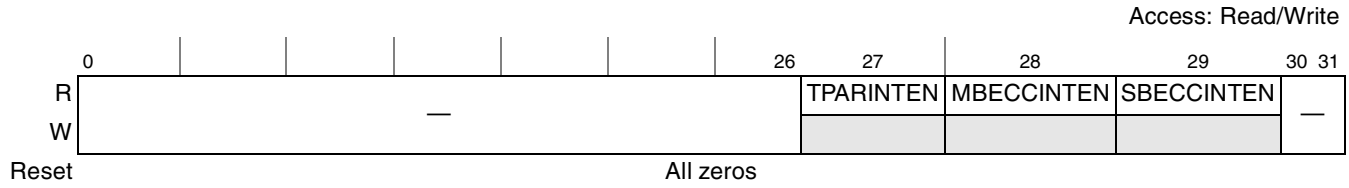


Figure 6-21. L2 Error Disable Register (L2ERRDIS)

Table 6-20 describes L2ERRINTEN fields.

Table 6-20. L2ERRINTEN Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27	TPARINTEN	Tag parity error reporting enable 0 Tag parity error reporting disabled 1 Tag parity error reporting enabled.
28	MBECCINTEN	Multiple-bit ECC error reporting enable 0 Multiple-bit ECC error reporting disabled 1 Multiple-bit ECC error reporting enabled
29	SBECCINTEN	Single-bit ECC error reporting enable 0 Single-bit ECC error reporting disabled 1 Single-bit ECC error reporting enabled
30–31	—	Reserved

6.1.6.5.11 L2 Error Attributes Capture Register (L2ERRATTR)

The L2 error attributes capture register (L2ERRATTR), shown in Figure 6-22, is a supervisor-level SPR that describes the L2 error attributes.

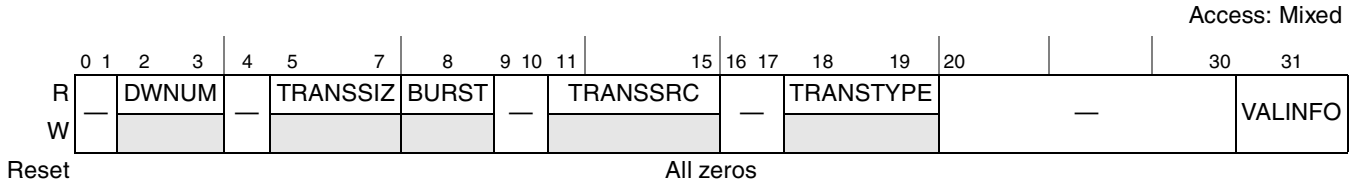


Figure 6-22. L2 Error Attributes Capture Register (L2ERRATTR)

Table 6-21 describes L2ERRATTR fields.

Table 6-21. L2ERRATTR Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–3	DWNUM	Double-word number of the detected error (data ECC errors only). Read only.
4	—	Reserved
5–7	TRANSSIZ	Transaction size for detected error (read only). Reserved for read transactions (TRANSTYPE = 10). The transaction size for a read to the L2 will always be a 32-byte burst. 000 8 bytes (single-beat) or reserved (burst) 001 1 byte (single-beat) or 16 bytes (burst) 010 2 bytes (single-beat) or 32 bytes (burst) 011 3 bytes (single beat) or reserved (burst) 100 4 bytes (single-beat) or reserved (burst) 101 5 bytes (single-beat) or reserved (burst) 110 6 bytes (single-beat) or reserved (burst) 111 7 bytes (single-beat) or reserved (burst)
8	BURST	Burst transaction for detected error. Read only. 0 Single-beat (≤ 64 bits) transaction 1 Burst transaction
9–10	—	Reserved
11–15	TRANSSRC	Transaction source for detected error. Read only. 00000External (logic external to the core) 10000Core (instruction) 10001Core (data)
16–17	—	Reserved
18–19	TRANSTYPE	Transaction type for detected error. Read only. 00 Snoop (tag/status read) 01 Write 10 Read 11 Reserved
20–30	—	Reserved
31	VALINFO	L2 capture registers valid 0 L2 capture registers contain no valid information or no enabled errors were detected. 1 L2 capture registers contain information of the first detected error that has reporting enabled. Software must clear this bit to unfreeze error capture so error detection hardware can overwrite the capture address/data/attributes for a newly detected error.

6.1.6.5.12 L2 Error Address Error Capture Register (L2ERRADDR)

The L2 error address error capture register (L2ERRADDR), shown in Figure 6-23, is a supervisor-level SPR that shows the L2 address corresponding to bits 4–35 of the detected error.

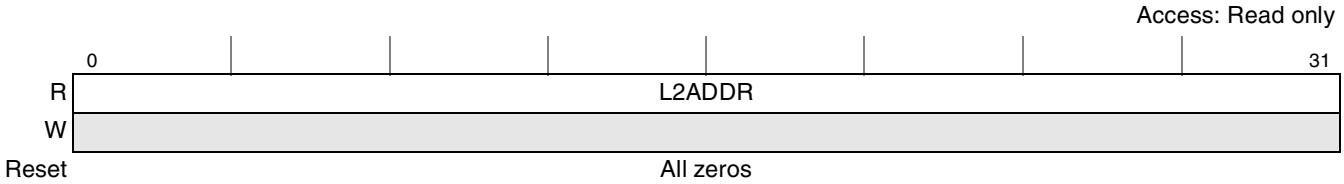


Figure 6-23. L2 Error Address Error Capture Register (L2ERRADDR)

Table 6-22 describes L2ERRADDR.

Table 6-22. L2ERRADDR Field Description

Bits	Name	Description
0–31	L2ADDR	L2 address[4:35] corresponding to detected error (read only)

6.1.6.5.13 L2 Error Address Error Capture Register (L2ERREADDR)

The L2 error address error capture register (L2ERREADDR), shown in Figure 6-24, is a supervisor-level SPR that shows the L2 address corresponding to bits 0–3 of the detected error.

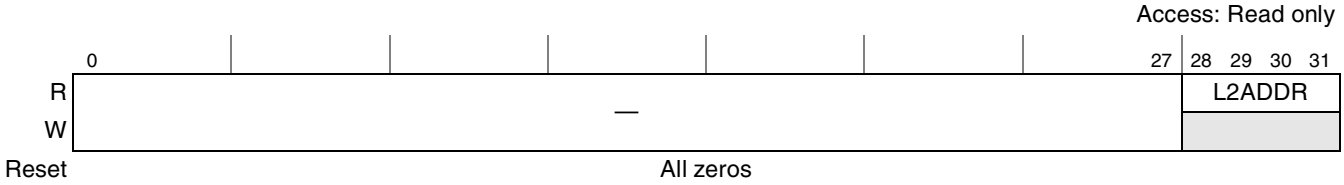


Figure 6-24. L2 Error Address Error Capture Register (L2ERREADDR)

Table 6-23 describes L2ERREADDR.

Table 6-23. L2ERREADDR Field Description

Bits	Name	Description
0–27	—	Reserved
28–31	L2EADDR	L2 address[0:3] corresponding to detected error (read only)

6.1.6.5.14 L2 Error Control Register (L2ERRCTL)

The L2 error control register (L2ERRCTL), shown in Figure 6-25, is a supervisor-level SPR that configures the L2 cache ECC error threshold and provides an L2 error count.

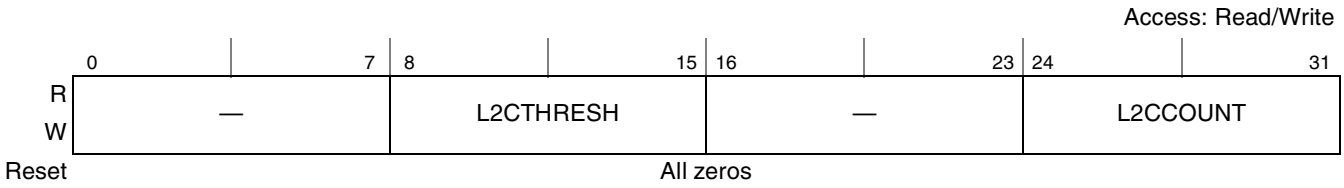


Figure 6-25. L2 Error Control Register (L2ERRCTL)

Table 6-24 describes L2ERRCTL fields.

Table 6-24. L2ERRCTL Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	L2CTHRESH	L2 cache threshold. Threshold value for the number of ECC single-bit errors that are detected before reporting an error condition.
16–23	—	Reserved
24–31	L2CCOUNT	L2 count. Counts ECC single-bit errors that have been detected. If L2CCOUNT equals the ECC single-bit error trigger threshold (L2CTHRESH), an error is reported if single-bit error reporting is enabled (SPECCDIS = 0). Software can write to this field.

6.1.6.5.15 Instruction Cache and Interrupt Control Register (ICTRL)

The instruction cache and interrupt control register (ICTRL), shown in Figure 6-26, is used in configuring interrupts and error reporting for the instruction and data caches. It is accessed as SPR 1011. Control and access to the ICTRL is through the privileged **mtspr**/**mfspr** instructions.

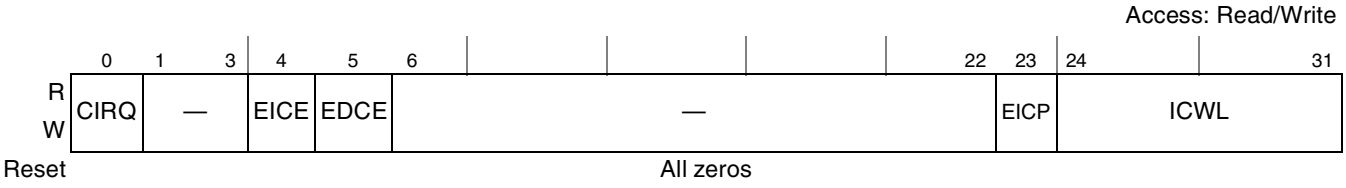


Figure 6-26. Instruction Cache and Interrupt Control Register (ICTRL)

Table 6-25 describes the bit fields for the ICTRL register.

Table 6-25. ICTRL Field Descriptions

Bits	Name	Description
0	CIRQ	CPU interrupt request 0 No processor interrupt request forwarded to interrupt handling. If software clears the CIRQ bit, it does not cancel a previously sent interrupt request. 1 Processor interrupt request sent to the interrupt mechanism. This interrupt request is combined with the external interrupt request (assertion of $\overline{int}$). When interrupts external to the core are enabled with the MSR[EE] bit and either this bit is set or $\overline{int}$ is asserted, the e600 core takes the external interrupt. If there is more than one interrupt request pending (CIRQ and $\overline{int}$ is asserted), only one interrupt is taken. When the external interrupt is taken, the ICTRL[CIRQ] bit is automatically cleared. Note that this mechanism allows a processor to interrupt itself. If software leaves CIRQ set while waiting for the interrupt to be taken, it can poll CIRQ to determine when the interrupt has been taken.
1–3	—	Reserved
4	EICE[1]	Instruction cache parity error enable 0 When the bit is cleared, any parity error in the L1 instruction cache is masked and does not cause machine checks or checkstop 1 Enables instruction cache parity error reporting. When an instruction cache parity error occurs, a machine check interrupt is taken if MSR[ME] = 1. When this condition occurs, SRR1[1] is set. For details on the machine check interrupt, see the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual*.
5	EDCE[2]	Data cache parity error enable 0 When the bit is cleared, any parity error in the L1 data cache is masked and does not cause machine checks or checkstop 1 Enables data cache parity error reporting. When a data cache parity error occurs, a machine check interrupt is taken if MSR[ME] = 1. When this condition occurs, SRR1[2] is set. For details on the machine check interrupt, see the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual*.
6–8	—	Reserved. Normally cleared. Used in debug. Writing nonzero values may cause boundedly undefined results.
9–22	—	Reserved. Read as zeros and ignores writes.
23	EICP	Enable instruction cache parity checking 0 Instruction cache parity disabled 1 When the EICP bit is set, the parity of any instructions fetched from the L1 instruction cache is checked. Any errors found are reported as instruction cache parity errors in SRR1. If EICE is also set, these instruction cache errors cause a machine check or checkstop. If either EICP or EICE is cleared, instruction cache parity is ignored. Note that when parity checking and error reporting are both enabled, errors are reported even on speculative fetches that are never actually executed. Correct instruction cache parity is always loaded into the L1 instruction cache regardless of whether checking is enabled or not.
24–31	ICWL[1]	Instruction cache way lock 0 Instruction cache way lock disabled. 1 Instruction cache way lock enabled. Each bit in ICWL corresponds to a way of the L1 instruction cache. Bit 24 corresponds to way 0, and bit 31 corresponds to way 7. Setting a bit locks the corresponding way in the instruction cache. Setting all 8 bits of ICWL is equivalent to locking the entire instruction cache. When all 8 ICWL bits are set, the e600 core behaves the same as when HID0[ILOCK] is set. See Section 6.1.6.1, “Hardware Implementation-Dependent Register 0 (HID0),” for details. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*. for suggestions on how to keep the PLRU replacement algorithm symmetrical, and for synchronization requirements for modifying ICWL.

[1] A context synchronizing instruction must precede and follow an mtspr.

[2] A dssall and sync must precede an mtspr and then a sync and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a dssall is not necessary prior to accessing the ICTRL[EDCE] bit.

ICTRL can be accessed with the **mtspr** and **mfspr** instructions using SPR 1011.

6.1.6.5.16 Load/Store Control Register (LDSTCR)

The load/store control register (LDSTCR) provides a way to lock the ways for the L1 data cache. The LDSTCR is shown in Figure 6-27.

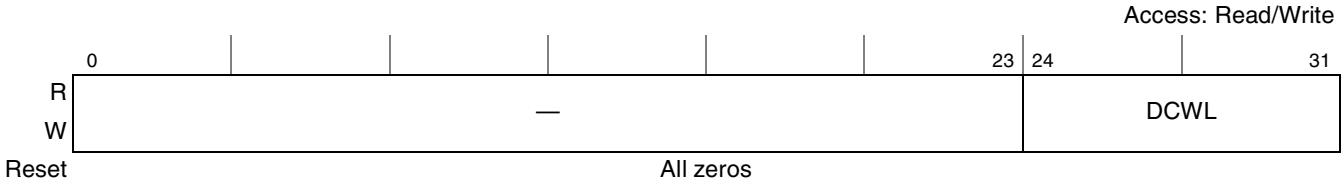


Figure 6-27. Load/Store Control Register (LDSTCR)

Table 6-30 describes the bit fields for the LDSTCR register.

Table 6-26. LDSTCR Field Descriptions

Bits	Name	Description
0–23	—	Reserved. Writing nonzero values may cause boundedly undefined results.
24–31	DCWL	Data cache way lock. Each bit in DCWL corresponds to a way of the L1 data cache. Bit 24 corresponds to way 0, and bit 31 corresponds to way 7. 0 Each cleared bit corresponds to the corresponding way not being locked in the L1 data cache. 1 Each set bit locks the corresponding way in the L1data cache. When DCWL[24–31] are all set, it is equivalent to locking the entire L1 data cache and the e600 core behaves the same as if HID0[DLOCK] is set. the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*. describes how to keep the PLRU replacement algorithm symmetrical and for more information on synchronization requirements with LDSTCR.

The LDSTCR register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1016. For synchronization requirements on the register see Section 6.3.2.4, “Synchronization.”

6.1.6.6 Instruction Address Breakpoint Register (IABR)

The instruction address breakpoint register (IABR), shown in Figure 6-28, supports the instruction address breakpoint interrupt. When this interrupt is enabled, instruction fetch addresses are compared with an effective address stored in the IABR. If the word specified in the IABR is fetched, the instruction breakpoint handler is invoked. The instruction that triggers the breakpoint does not execute before the handler is invoked. For more information, see the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual*. The IABR can be accessed with **mtspr** and **mfspr** using SPR 1010. The e600 core requires that an **mtspr**[IABR] be followed by a context synchronizing instruction. The e600 core may not generate a breakpoint response for that context synchronizing instruction if the breakpoint was enabled by **mtspr**[IABR] immediately preceding it. The e600 core cannot block a breakpoint response on the context

synchronizing instruction if the breakpoint was disabled by **mtspr**[IABR] immediately preceding it. For more information on synchronization see Section 6.3.2.4.1, “Context Synchronization.”

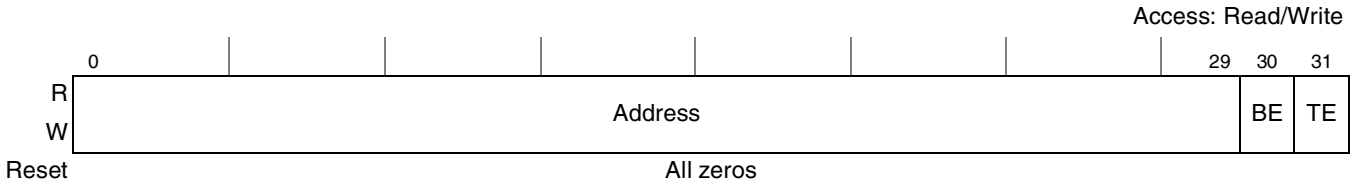


Figure 6-28. Instruction Address Breakpoint Register (IABR)

The IABR bits are described in Table 6-27.

Table 6-27. IABR Field Descriptions

Bits[1]	Name	Description
0–29	Address	Word instruction breakpoint address to be compared with EA[0–29] of the next instruction
30	BE	Breakpoint enabled. Setting this bit enables breakpoint address checking.
31	TE	Translation enable IABR[TE] must equal MSR[IR] in order for a match to be signaled. When IABR[TE] and MSR[IR] = 0 or when IABR[TE] and MSR[IR] = 1, a match is signaled.

1 A context synchronizing instruction must follow an mtspr.

6.1.6.7 Memory Management Registers Used for Software Table Searching

This section describes the registers used by the e600 core when software table searching is enabled (HID0[STEN] = 1) and a TLB miss interrupt occurs. Software table searching is described in detail in the “Memory Management” chapter of the *e600 PowerPC Core Reference Manual.*

6.1.6.7.1 TLB Miss Register (TLBMISS)

The TLBMISS register is automatically loaded by the e600 core when software table searching is enabled (HID0[STEN] = 1) and a TLB miss interrupt occurs. Its contents are used by the TLB miss interrupt handlers (the software table search routines) to start the search process. Note that the e600 core always loads a big-endian address into the TLBMISS register. This register is read-only. The TLBMISS register has the format shown in Figure 6-29 for the e600 core.

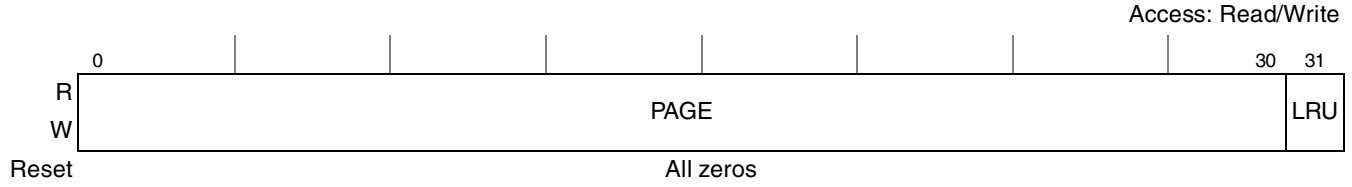


Figure 6-29. TLBMISS Register (TLBMISS)

Table 6-28 describes the bits in the TLBMISS register.

Table 6-28. TLBMISS Register—Field and Bit Descriptions

Bits	Name	Description
0–30	PAGE	Effective page address. Stores EA[0–30] of the access that caused the TLB miss interrupt.
31	LRU	Least recently used way of the addressed TLB set. The LRU bit can be loaded into bit 31 of **r**B, prior to execution of **tlbli** or **tlbld** to select the way to be replaced for a TLB miss. However, this value should be inverted in **r**B prior to execution of **tlbli** or **tlbld** for a TLB miss interrupt caused by the need to update the C-bit.

TLBMISS can be accessed with **mtspr** and **mfspr** using SPR 980.

6.1.6.7.2 Page Table Entry Registers (PTEHI and PTELO)

The PTEHI and PTELO registers are used by the **tlbld** and **tlbli** instructions to create a TLB entry. When software table searching is enabled (HID0[STEN] = 1) and a TLB miss interrupt occurs, the bits of the page table entry (PTE) for this access are located by software and saved in the PTE registers. Figure 6-30 and Figure 6-31 show the format for two supervisor registers, PTEHI and PTELO, respectively.

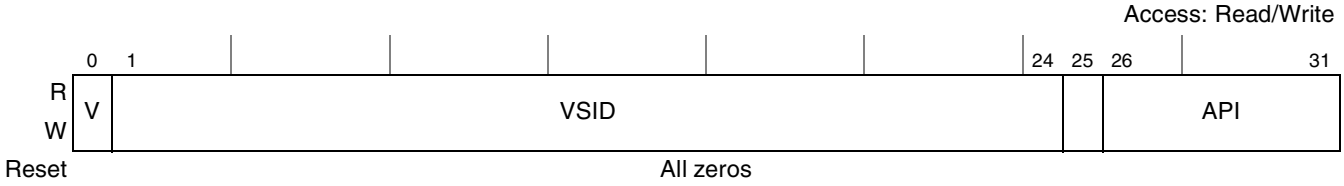


Figure 6-30. PTEHI Register—Extended Addressing

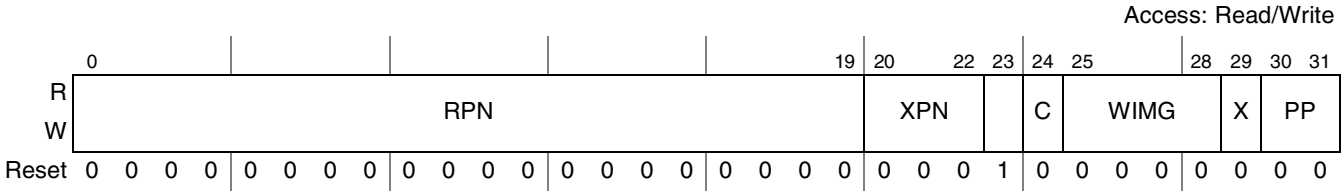


Figure 6-31. PTELO Register—Extended Addressing

Note that the contents of PTEHI are automatically loaded when any of the three software table search interrupts is taken. PTELO is loaded by the software table search routines (the TLB miss interrupt handlers) based on the valid PTE located in the page tables prior to execution of the **tlbli** or **tlbld** instruction.

Table 6-29 lists the corresponding bit definitions for the PTEHI and PTELO registers.

Table 6-29. PTEHI and PTELO Bit Definitions

Register	Bit	Name	Description
PTEHI	0	V	Entry valid (V = 1) or invalid (V = 0). Always set by the processor on a TLB miss interrupt.
	1–24	VSID	Virtual segment ID. The corresponding SR[VSID] field is copied to this field.
	25	—	Reserved. Corresponds to the hash function identifier in PTE.
	26–31	API	Abbreviated page index. TLB miss interrupts will set this field with bits from TLBMISS[4–9] which are bits from the effective address for the access that caused the software table search operation. The **tlbld** and **tlbli** instructions ignore the API bits in PTEHI register and get the API from the instruction's operand**,** **r**B. However, for future compatibility, the API in **r**B should match the PTEHI[API].
PTELO	0–19	RPN	Physical page number
	20–22	XPN	Extended page number The XPN field provides the physical address bits, PA[0–2].
	23	—	Reserved. Corresponds to the reference bit in a PTE. The referenced bit is not stored in the page tables, so this bit is ignored in the PTELO register. To simplify software, this bit is hard-wired to 1. All the other bits in PTELO correspond to the bits in the low word of the PTE.
	24	C	Changed bit
	25–28	WIMG	Memory/cache control bits
	29	X	Extended page number The X field provides the physical address bit 3, PA[3].
	30–31	PP	Page protection bits

When extended addressing is not enabled, (HID0[XAEN] = 0), the software must clear the PTELO[XPN] and PTELO[X] bits; otherwise whatever values are in the fields become the four most-significant bits of the physical address. **Note:** The PTEHI register is accessed with **mtspr** and **mfspr** as SPR 981 and PTELO is accessed as SPR 982.

6.1.6.8 Thermal Management Register

The e600 core provides an instruction cache throttling mechanism to reduce the instruction execution rate without the complexity and overhead of dynamic clock control. When used with dynamic power management, instruction cache throttling provides the system designer with a flexible way to control device temperature while allowing the processor to continue operating. Note that performance does degrade when instruction cache throttling is enabled to reduce junction temperature; entering short bursts of nap or sleep is a superior method for reducing thermal output and power consumption.

6.1.6.8.1 Instruction Cache Throttling Control Register (ICTC)

Reducing the rate of instruction fetching can control junction temperature without the complexity and overhead of dynamic clock control. System software can control instruction forwarding by writing a nonzero value to the ICTC register, a supervisor-level register shown in Figure 6-32. The overall junction

temperature reduction comes from the dynamic power management of each functional unit when the e600 core is idle in between instruction fetches.

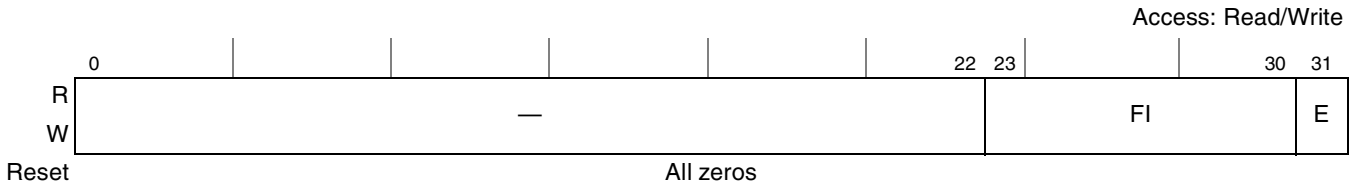


Figure 6-32. Instruction Cache Throttling Control Register (ICTC)

Table 6-30 describes the bit fields for the ICTC register.

Table 6-30. ICTC Field Descriptions

Bits	Name	Description
0–22	—	Reserved. The bits should be cleared.
23–30	INTERVAL	Instruction forwarding interval expressed in core clocks. When throttling is enabled, the interval field specifies the minimum number of cycles between instructions being dispatched. (The core dispatches one instruction every INTERVAL cycle.) The minimum interval for throttling control is 2 cycles. 0x00, 0x01, 0x02 One instruction dispatches every 2 core clocks. 0x03 One instruction dispatches every 3 core clocks ... 0xFF One instruction dispatches every 255 core clocks.
31	E	Enable instruction throttling 0 Instructions dispatch normally. 1 Only one instruction dispatches every INTERVAL cycles.

Instruction cache throttling is enabled by setting ICTC[E] and writing the instruction forwarding interval into ICTC[INTERVAL]. A context synchronizing instruction should be executed after a move to the ICTC register to ensure that it has taken effect. Enabling, disabling, or changing the instruction forwarding interval affects instruction forwarding immediately.

The ICTC register can be accessed with the **mtspr** and **mfspr** instructions using SPR 1019.

6.1.6.9 Performance Monitor Registers

This section describes the registers used by the performance monitor, which is described in the "Performance Monitor" chapter of the *e600 PowerPC Core Reference Manual*.

6.1.6.9.1 Monitor Mode Control Register 0 (MMCR0)

The monitor mode control register 0 (MMCR0), shown in Figure 6-33, is a 32-bit SPR provided to specify events to be counted and recorded. If the state of MSR[PR] and MSR[PMM] matches a state specified in MMCR0, then counting is enabled. See the "Performance Monitor" chapter of the *e600 PowerPC Core Reference Manual* for further details. The MMCR0 can be accessed only in supervisor mode. User-level software can read the contents of MMCR0 by issuing an **mfspr** instruction to UMMCR0, described in Section 6.1.6.9.2, "User Monitor Mode Control Register 0 (UMMCR0)."

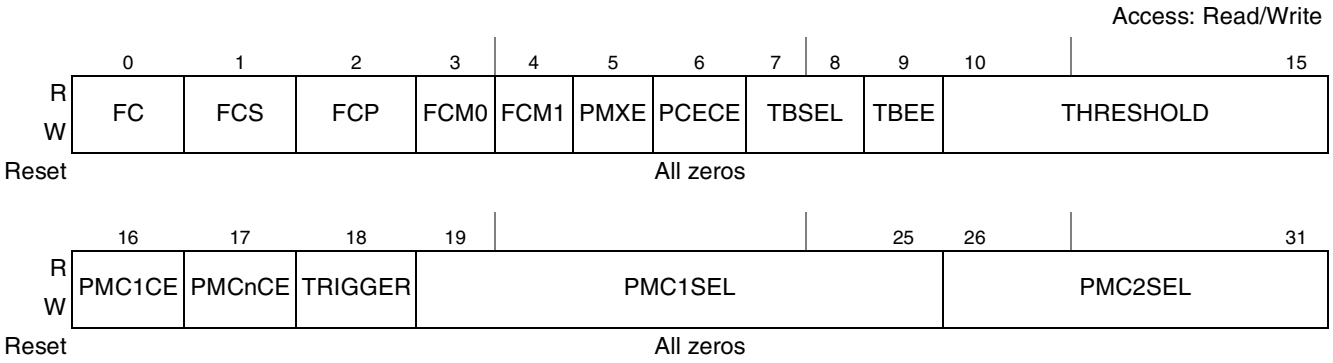


Figure 6-33. Monitor Mode Control Register 0 (MMCR0)

This register is automatically cleared at power-up. Reading this register does not change its contents. Table 6-31 describes MMCR0 fields.

Table 6-31. MMCR0 Field Descriptions

Bits	Name	Description
0	FC	Freeze counters 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented (performance monitor counting is disabled). The processor sets this bit when an enabled condition or event occurs and MMCR0[FCECE] = 1. Note that SIAR is not updated if performance monitor counting is disabled.
1	FCS	Freeze counters in supervisor mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 0.
2	FCP	Freeze counters in user mode 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PR] = 1.
3	FCM1	Freeze counters while mark = 1 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 1.
4	FCM0	Freeze counters while mark = 0 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are not incremented if MSR[PMM] = 0.
5	PMXE	Performance monitor interrupt enable 0 Performance monitor interrupts are disabled. 1 Performance monitor interrupts are enabled until a performance monitor interrupt occurs, at which time MMCR0[PMXE] is cleared. Software can clear PMXE to prevent performance monitor interrupts. Software can also set PMXE and then poll it to determine whether an enabled condition or event occurred.
6	FCECE	Freeze counters on enabled condition or event 0 The PMCs are incremented (if permitted by other MMCR bits). 1 The PMCs are incremented (if permitted by other MMCR bits) until an enabled condition or event occurs when MMCR0[TRIGGER] = 0, at which time MMCR0[FC] is set. If the enabled condition or event occurs when MMCR0[TRIGGER] = 1, FCECE is treated as if it were 0. The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in the “Performance Monitor” chapter of the *e600 PowerPC Core Reference Manual*.

Table 6-31. MMCR0 Field Descriptions (continued)

Bits	Name	Description
7–8	TBSEL	Time base selector. Selects the time base bit that can cause a time base transition event (the event occurs when the selected bit changes from 0 to 1). 00 TBL[31] 01 TBL[23] 10 TBL[19] 11 TBL[15] Time base transition events can be used to periodically collect information about processor activity. In multiprocessor systems in which the TB registers are synchronized among processors, time base transition events can be used to correlate the performance monitor data obtained by the several processors. For this use, software must specify the same TBSEL value for all the processors in the system. Because the time-base frequency is implementation-dependent, software should invoke a system service program to obtain the frequency before choosing a value for TBSEL.
9	TBEE	Time base event enable 0 Time-base transition events are disabled. 1 Time-base transition events are enabled. A time-base transition is signaled to the performance monitor if the TB bit specified in MMCR0[TBSEL] changes from 0 to 1. Time-base transition events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an interrupt (MMCR0[PMXE]). Changing the bits specified in MMCR0[TBSEL] while MMCR0[TBEE] is enabled may cause a false 0 to 1 transition that signals the specified action (freeze, trigger, or interrupt) to occur immediately.
10–15	THRESHOLD	Threshold. Contains a threshold value between 0 to 63. Two types of thresholds can be counted. The first type counts any event that lasts longer than the threshold value and uses MMCR2[THRESHMULT] to scale the threshold value by 2 or 32. The second type counts only the events that exceed the threshold value. This type does not use MMCR2[THRESHMULT] to scale the threshold value. By varying the threshold value, software can obtain a profile of the characteristics of the events subject to the threshold. For example, if PMC1 counts cache misses for which the duration exceeds the threshold value, software can obtain the distribution of cache miss durations for a given program by monitoring the program repeatedly using a different threshold value each time.
16	PMC1CE	PMC1 condition enable. Controls whether counter negative conditions due to a negative value in PMC1 are enabled. 0 Counter negative conditions for PMC1 are disabled. 1 Counter negative conditions for PMC1 are enabled. Such events can be used to freeze counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an interrupt (MMCR0[PMXE]).
17	PMC*n*CE	PMC*n* condition enable. Controls whether counter negative conditions due to a negative value in any PMC*n* (that is, in any PMC except PMC1) are enabled. 0 Counter negative conditions for all PMC*n*s are disabled. 1 Counter negative conditions for all PMC*n*s are enabled. These events can be used to freeze the counters (MMCR0[FCECE]), trigger the counters (MMCR0[TRIGGER]), or signal an interrupt (MMCR0[PMXE]).

Table 6-31. MMCR0 Field Descriptions (continued)

Bits	Name	Description
18	TRIGGER	Trigger 0 The PMCs are incremented (if permitted by other MMCR bits). 1 PMC1 is incremented (if permitted by other MMCR bits). The PMC*n*s are not incremented until PMC1 is negative or an enabled timebase or event occurs, at which time the PMC*n*s resume incrementing (if permitted by other MMCR bits) and MMCR0[TRIGGER] is cleared. The description of FCECE explains the interaction between TRIGGER and FCECE. Uses of TRIGGER include the following: • Resume counting in the PMC*n*s when PMC1 becomes negative without causing a performance monitor interrupt. Then freeze all PMCs (and optionally cause a performance monitor interrupt) when a PMC*n* becomes negative. The PMC*n*s then reflect the events that occurred after PMC1 became negative and before PMC*n* becomes negative. This use requires the following MMCR0 bit settings. – TRIGGER = 1 – PMC1CE = 0 – PMC*n*CE = 1 – TBEE = 0 – FCECE = 1 – PMXE = 1 (if a performance monitor interrupt is desired) • Resume counting in the PMC*n*s when PMC1 becomes negative, and cause a performance monitor interrupt without freezing any PMCs. The PMC*n*s then reflect the events that occurred between the time PMC1 became negative and the time the interrupt handler reads them. This use requires the following MMCR0 bit settings. – TRIGGER = 1 – PMC1CE = 1 – TBEE = 0 – FCECE = 0 – PMXE = 1 The use of the trigger and freeze counter conditions depends on the enabled conditions and events described in the “Performance Monitor” chapter of the *e600 PowerPC Core Reference Manual*.
19–25	PMC1SEL	PMC1 selector. Contains a code (one of at most 128 values) that identifies the event to be counted in PMC1. See the “Performance Monitor” chapter of the *e600 PowerPC Core Reference Manual*.
26–31	PMC2SEL	PMC2 selector. Contains a code (one of at most 64 values) that identifies the event to be counted in PMC2. See the “Performance Monitor” chapter of the *e600 PowerPC Core Reference Manual*.

MMCR0 can be accessed with **mtspr** and **mfspr** using SPR 952.

6.1.6.9.2 User Monitor Mode Control Register 0 (UMMCR0)

The contents of MMCR0 are reflected to UMMCR0, which can be read by user-level software. MMCR0 can be accessed with **mfspr** using SPR 936.

6.1.6.9.3 Monitor Mode Control Register 1 (MMCR1)

The monitor mode control register 1 (MMCR1) functions as an event selector for performance monitor counter registers 3, 4, 5, and 6 (PMC3, PMC4, PMC5, PMC6). The MMCR1 register is shown in Figure 6-34.

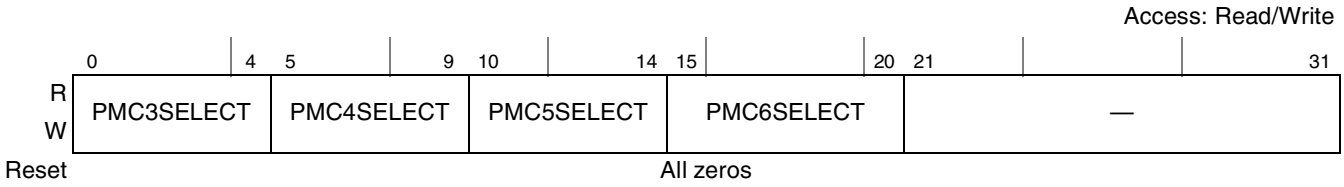


Figure 6-34. Monitor Mode Control Register 1 (MMCR1)

Bit settings for MMCR1 are shown in Table 6-32. The corresponding events are described in Section 6.1.6.9.8, "Performance Monitor Counter Registers (PMC1–PMC6)."

Table 6-32. MMCR1 Field Descriptions

Bits	Name	Description
0–4	PMC3SELECT	PMC3 selector. Contains a code (1 of at most 32 values) that identifies the event to be counted in PMC3. See the "Performance Monitor" chapter of the *e600 PowerPC Core Reference Manual*.
5–9	PMC4SELECT	PMC4 selector. Contains a code (1 of at most 32 values) that identifies the event to be counted in PMC4. See the "Performance Monitor" chapter of the *e600 PowerPC Core Reference Manual*.
10–14	PMC5SELECT	PMC5 selector. Contains a code (1 of at most 32 values) that identifies the event to be counted in PMC5. See the "Performance Monitor" chapter of the *e600 PowerPC Core Reference Manual*.
15–20	PMC6SELECT	PMC6 selector. Contains a code (1 of at most 64 values) that identifies the event to be counted in PMC6. See the "Performance Monitor" chapter of the *e600 PowerPC Core Reference Manual*.
21–31	—	Reserved

MMCR1 can be accessed with **mtspr** and **mfspr** using SPR 956. User-level software can read the contents of MMCR1 by issuing an **mfspr** instruction to UMMCR1, described in Section 6.1.6.9.4, "User Monitor Mode Control Register 1 (UMMCR1)."

6.1.6.9.4 User Monitor Mode Control Register 1 (UMMCR1)

The contents of MMCR1 are reflected to UMMCR1, which can be read by user-level software. MMCR1 can be accessed with **mfspr** using SPR 940.

6.1.6.9.5 Monitor Mode Control Register 2 (MMCR2)

The monitor mode control register 2 (MMCR2) functions as an event selector for performance monitor counter registers 3 and 4 (PMC3 and PMC4). The MMCR2 register is shown in Figure 6-35.

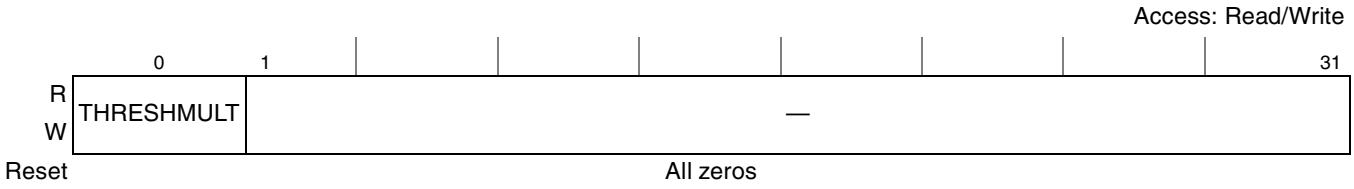


Figure 6-35. Monitor Mode Control Register 2 (MMCR2)

Table 6-33 describes MMCR2 fields.

Table 6-33. MMCR2 Field Descriptions

Bits	Name	Description
0	THRESHMULT	Threshold multiplier. Used to extend the range of the THRESHOLD field, MMCR0[10–15]. 0 Threshold field is multiplied by 2 1 Threshold field is multiplied by 32
1–31	—	Reserved

MMCR2 can be accessed with **mtspr** and **mfspr** using SPR 944. User-level software can read the contents of MMCR2 by issuing an **mfspr** instruction to UMMCR2, described in Section 6.1.6.9.6, “User Monitor Mode Control Register 2 (UMMCR2).”

6.1.6.9.6 User Monitor Mode Control Register 2 (UMMCR2)

The contents of MMCR2 are reflected to UMMCR2, which can be read by user-level software. UMMCR2 can be accessed with the **mfspr** instruction using SPR 928.

6.1.6.9.7 Breakpoint Address Mask Register (BAMR)

The breakpoint address mask register (BAMR), shown in Figure 6-36, is used in conjunction with the events that monitor IABR hits.

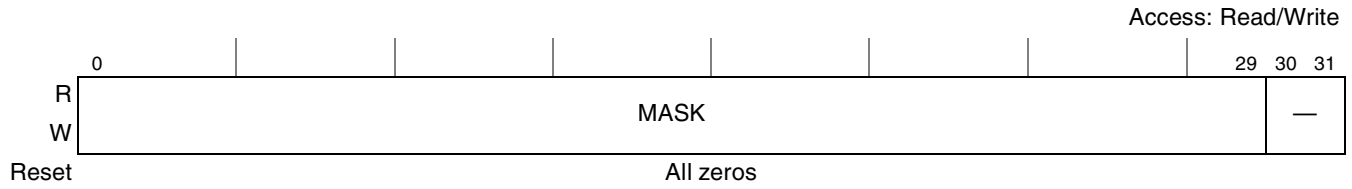


Figure 6-36. Breakpoint Address Mask Register (BAMR)

Table 6-34 describes BAMR fields.

Table 6-34. BAMR Field Descriptions

Bit	Name	Description
0–29	MASK[1]	Used with PMC1 event (PMC1 event 42) that monitor IABR hits. The addresses to be compared for an IABR match are affected by the value in BAMR: • IABR hit (PMC1, event 42) occurs if IABR_CMP (that is, IABR AND BAMR) = instruction_address_compare (that is, EA AND BAMR) IABR_CMP[0–29] = IABR[0–29] AND BAMR[0–29] instruction_addr_cmp[0–29] = instruction_addr[0–29] AND BAMR[0–29] Be aware that breakpoint event 42 of PMC1 can be used to trigger performance monitor interrupts when the performance monitor detects an enabled overflow. This feature supports debug purposes and occurs only when IABR[30] is set. To avoid taking one of the above interrupts, make sure that IABR[30] is cleared.
30–31	—	Reserved

[1] A context synchronizing instruction must follow the mtspr.

BAMR can be accessed with **mtspr** and **mfspr** using SPR 951. For synchronization requirements on the register see Section 6.3.2.4, “Synchronization.”

6.1.6.9.8 Performance Monitor Counter Registers (PMC1–PMC6)

PMC1–PMC6, shown in Figure 6-37, are 32-bit counters that can be programmed to generate a performance monitor interrupt when they overflow.

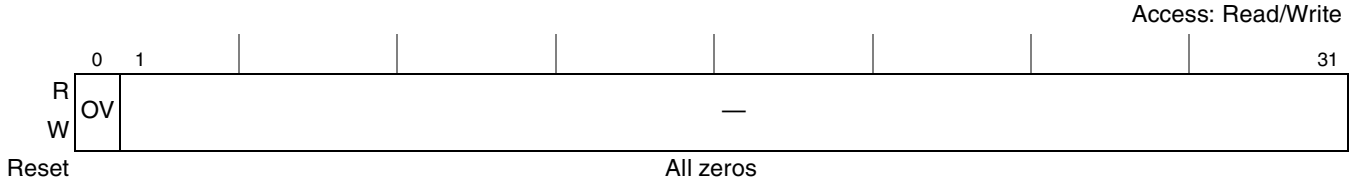


Figure 6-37. Performance Monitor Counter Registers (PMC1–PMC6)

The bits contained in the PMC registers are described in Table 6-35.

Table 6-35. PMC*n* Field Descriptions

Bits	Name	Description
0	OV	Overflow When this bit is set, it indicates that this counter has overflowed and reached its maximum value so that PMCn[OV] = 1.
1–31	Counter value	Counter value Indicates the number of occurrences of the specified event.

Counters overflow when the high-order (sign) bit becomes set; that is, they reach the value 2,147,483,648 (0x8000_0000). However, an interrupt is not generated unless both MMCR0[PMXE] and either MMCR0[PMC1CE] or MMCR0[PMC*c*CE] are also set as appropriate.

Note that the interrupt can be masked by clearing MSR[EE]; the performance monitor condition may occur with MSR[EE] cleared, but the interrupt is not taken until MSR[EE] is set. Setting MMCR0[FCECE] forces counters to stop counting when a counter interrupt or any enabled condition or event occurs. Setting MMCR0[TRIGGER] forces counters PMC*n* (*n* > 1), to begin counting when PMC1 goes negative or an enabled condition or event occurs.

Software is expected to use the **mtspr** instruction to explicitly set PMC to non-overflowed values. Setting an overflowed value may cause an erroneous interrupt. For example, if both MMCR0[PMXE] and either MMCR0[PMC1CE] or MMCR0[PMC*n*CE] are set and the **mtspr** instruction loads an overflow value, an interrupt may be taken without an event counting having taken place.

The PMC registers can be accessed with the **mtspr** and **mfspr** instructions using the following SPR numbers:

- PMC1 is SPR 953
- PMC2 is SPR 954
- PMC3 is SPR 957
- PMC4 is SPR 958
- PMC5 is SPR 945
- PMC6 is SPR 946

6.1.6.9.9 User Performance Monitor Counter Registers (UPMC1–UPMC6)

The contents of the PMC1–PMC6 are reflected to UPMC1–UPMC6, which can be read by user-level software. The UPMC registers can be read with **mfspr** using the following SPR numbers:

- UPMC1 is SPR 937
- UPMC2 is SPR 938
- UPMC3 is SPR 941
- UPMC4 is SPR 942
- UPMC5 is SPR 929
- UPMC6 is SPR 930

6.1.6.9.10 Sampled Instruction Address Register (SIAR)

The sampled instruction address register (SIAR) is a supervisor-level register that contains the effective address of the last instruction to complete before the performance monitor interrupt is signaled. The SIAR is shown in Figure 6-38.

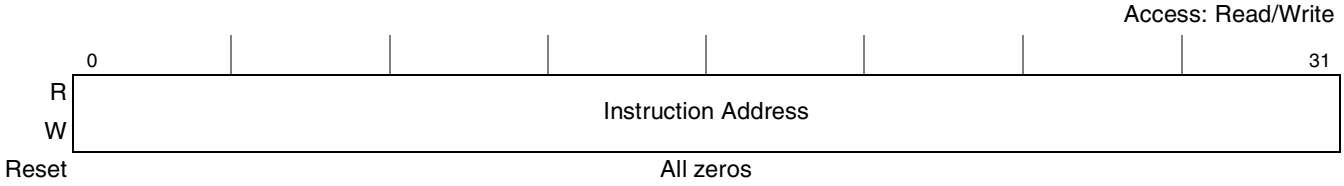


Figure 6-38. Sampled Instruction Address Registers (SIAR)

Note that SIAR is not updated in any of the following conditions:

- Performance monitor counting has been disabled by setting MMCR0[FC].
- Performance monitor interrupt has been disabled by clearing MMCR0[PMXE]

SIAR can be accessed with the **mtspr** and **mfspr** instructions using SPR 955.

6.1.6.9.11 User-Sampled Instruction Address Register (USIAR)

The contents of SIAR are reflected to USIAR, which can be read by user-level software. USIAR can be accessed with the **mfspr** instructions using SPR 939.

6.1.6.9.12 Sampled Data Address Register (SDAR) and User-Sampled Data Address Register (USDAR)

The e600 core does not implement the sampled data address register (SDAR) or the user-level, read-only USDA registers. Note that in previous processors the SDAR and USDAR registers could be written to by boot code without causing an interrupt, this is not the case in the e600 core. A **mtspr** or **mfspr** SDAR or USDAR instruction causes a program interrupt.

6.1.7 Reset Settings

Table 6-36 shows the state of the registers and other resources after a hard reset and before the first instruction is fetched from address 0xFFF0_0100 (the system reset interrupt vector). When a register is not initialized at hard reset, the setting is undefined.

Table 6-36. Settings Caused by Hard Reset (Used at Power-On)

Resource	Setting
BAMR	All zeros
BATs	Undefined
Caches (L1/L2)	Disabled. The caches are not invalidated and must be invalidated in software before they are enabled.
CR	All zeros
CTR	All zeros
DABR	Breakpoint is disabled. Address is undefined.
DAR	All zeros
DEC	0xFFFF_FFFF
DSISR	All zeros
EAR	All zeros
FPRs	Undefined
FPSCR	All zeros
GPRs	Undefined
HID0	0x8000_0000

Table 6-36. Settings Caused by Hard Reset (Used at Power-On) (continued)

Resource	Setting
HID1	0x000*n*_*n*080 (Note that bits 14–19 are set to match the settings of *pll_cfg*[0:5] at reset. See "Chapter 4, "Reset, Clocking, and Initialization," for more information.)
IABR	All zeros (breakpoint is disabled)
ICTC	All zeros
ICTRL	All zeros
L2CAPTDATAHI	All zeros
L2CAPTDATALO	All zeros
L2CAPTECC	All zeros
L2CR	0x1000_0000
L2ERRADDR	All zeros
L2ERRATTR	All zeros
L2ERRCTL	All zeros
L2ERRDET	All zeros
L2ERRDIS	All zeros
L2ERREADDR	All zeros
L2ERRINJCTL	All zeros
L2ERRINJHI	All zeros
L2ERRINJLO	All zeros
L2ERRINTEN	All zeros
LDSTCR	All zeros
LR	All zeros
MMCR*n*	All zeros
MSSCR0	0x0000_8000
MSR	0x0000_0040 (only IP set)
S	All zeros
PMC*n*	Undefined
PTEHI	All zeros
PTELO	All zeros
PVR	0x8004_*nnnn*
Reservation address	Undefined
Reservation flag	Cleared
SDR1	All zeros
SIAR	All zeros

Table 6-36. Settings Caused by Hard Reset (Used at Power-On) (continued)

Resource	Setting
SPRG0–SPGR7	All zeros
SRs	Undefined
SRR0	All zeros
SRR1	All zeros
TBU and TBL	All zeros
TLBs	Undefined
TLBMISS	All zeros
UMMCR*n*	All zeros
UPMC*n*	All zeros
USIAR	All zeros
VRs	Undefined
VRSAVE	All zeros
VSCR	0x0001_0000
XER	All zeros

6.2 Operand Conventions

This section describes the operand conventions as they are represented in two levels of the PowerPC architecture—UISA and VEA. Detailed descriptions are provided of conventions used for storing values in registers and memory, accessing PowerPC registers, and representation of data in these registers.

6.2.1 Floating-Point Execution Models—UISA

The IEEE Std. 754 standard defines conventions for 64- and 32-bit arithmetic. The standard requires that single-precision arithmetic be provided for single-precision operands. The standard permits double-precision arithmetic instructions to have either (or both) single-precision or double-precision operands but states that single-precision arithmetic instructions should not accept double-precision operands.

The PowerPC UISA follows these guidelines:

- Double-precision arithmetic instructions can have single-precision operands but always produce double-precision results.
- Single-precision arithmetic instructions require all operands to be single-precision and always produce single-precision results.

For arithmetic instructions, conversion from double- to single-precision must be done explicitly by software, while conversion from single- to double-precision is done implicitly by the processor.

All implementations of the PowerPC architecture provide the equivalent of the following execution models to ensure that identical results are obtained. The definition of the arithmetic instructions for infinities, denormalized numbers, and NaNs follow conventions described in the following sections.

Although the double-precision format specifies an 11-bit exponent, exponent arithmetic uses two additional bit positions to avoid potential transient overflow conditions. An extra bit is required when denormalized double-precision numbers are prenormalized. A second bit is required to permit computation of the adjusted exponent value in the following examples when the corresponding interrupt enable bit has a value of 1:

- Underflow during multiplication using a denormalized operand
- Overflow during division using a denormalized divisor

6.2.2 Data Organization in Memory and Data Transfers

Bytes in memory are numbered consecutively starting with 0. Each number is the address of the corresponding byte.

Memory operands can be bytes, half-words, words, double-words, quad-words, or, for the load/store multiple and load/store string instructions, a sequence of bytes or words. The address of a memory operand is the address of its first byte (that is, of its lowest-numbered byte). Operand length is implicit for each instruction.

6.2.3 Alignment and Misaligned Accesses

The operand of a single-register memory access instruction has an alignment boundary equal to its length. An operand’s address is misaligned if it is not a multiple of its width.

The concept of alignment is also applied more generally to data in memory. For example, a 12-byte data item is said to be word-aligned if its address is a multiple of four.

Some instructions require their memory operands to have certain alignment. In addition, alignment can affect performance. For single-register memory access instructions, the best performance is obtained when memory operands are aligned.

Instructions are 32 bits (one word) long and must be word-aligned.

The e600 core does not provide hardware support for floating-point memory that is not word-aligned. If a floating-point operand is not word-aligned, the e600 core invokes an alignment interrupt, and it is left up to software to break up the offending memory access operation appropriately. In addition, some non-double-word–aligned memory accesses suffer performance degradation as compared to an aligned access of the same type.

In general, floating-point word accesses should always be word-aligned and floating-point double-word accesses should always be double-word–aligned. Frequent use of misaligned accesses is discouraged because they can degrade overall performance.

6.2.4 Floating-Point Operands

The e600 core provides hardware support for all single- and double-precision floating-point operations for most value representations and all rounding modes. This architecture provides for hardware to implement a floating-point system as defined in ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating Point Arithmetic*. Detailed information about the floating-point execution model can be found in Chapter 3, “Operand Conventions,” in the *Programming Environments Manual*.

The e600 core supports non-IEEE Std. 754 mode when FPSCR[29] is set. In this mode, denormalized numbers are treated in a non-IEEE Std. 754 conforming manner. This is accomplished by delivering results that are forced to the value zero.

6.3 Instruction Set Summary

This chapter describes instructions and addressing modes defined for the e600 core. These instructions are divided into the following functional categories:

- Integer instructions—These include arithmetic and logical instructions. For more information, see Section 6.3.4.1, “Integer Instructions.”
- Floating-point instructions—These include floating-point arithmetic instructions, as well as instructions that affect the floating-point status and control register (FPSCR). For more information, see Section 6.3.4.2, “Floating-Point Instructions.”

Load and store instructions—These include integer and floating-point load and store instructions. For more information, see Section 6.3.4.3, “Load and Store Instructions.”

- Flow control instructions—These include branching instructions, condition register logical instructions, trap instructions, and other instructions that affect the instruction flow. For more information, see Section 6.3.4.4, “Branch and Flow Control Instructions.”
- Processor control instructions—These instructions are used for synchronizing memory accesses and managing segment registers. For more information, see Section 6.3.4.6, “Processor Control Instructions—UISA,” Section 6.3.5.1, “Processor Control Instructions—VEA,” and Section 6.3.6.2, “Processor Control Instructions—OEA.”
- Memory synchronization instructions—These instructions are used for memory synchronizing. See Section 6.3.4.7, “Memory Synchronization Instructions—UISA,” and Section 6.3.5.2, “Memory Synchronization Instructions—VEA,” for more information.
- Memory control instructions—These instructions provide control of caches and TLBs. For more information, see Section 6.3.5.3, “Memory Control Instructions—VEA,” and Section 6.3.6.3, “Memory Control Instructions—OEA.”
- External control instructions—These include instructions for use with special input/output devices. For more information, see Section 6.3.5.4, “Optional External Control Instructions.”
- AltiVec instructions–AltiVec technology does not have optional instructions defined, so all instructions listed in the *AltiVec Technology Programming Environments Manual* are implemented for the e600 core. Instructions that are implementation-specific are described in Section 6.6.2, “AltiVec Instructions with Specific Implementations for the e600 Core.”

Note that this grouping of instructions does not necessarily indicate the execution unit that processes a particular instruction or group of instructions. This information, which is useful for scheduling instructions most effectively, is provided in the “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual*.

Integer instructions operate on word operands. Floating-point instructions operate on single-precision and double-precision floating-point operands. AltiVec instructions operate on byte, half-word, word, and quad-word 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 general-purpose registers (GPRs). It provides for word and double-word operand loads and stores between memory and a set of 32 floating-point registers (FPRs). It also provides for byte, half-word, word, and quad-word operand loads and stores between memory and a set of 32 vector registers (VRs).

Arithmetic and logical instructions do not read or modify memory. To use the contents of a memory location 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 to the target location using load and store instructions.

The description of each instruction includes the mnemonic and a formatted list of operands. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for some of the frequently-used instructions; see Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for a complete list of simplified mnemonics. Programs written to be portable across the various assemblers for the PowerPC architecture should not assume the existence of mnemonics not described in that document.

6.3.1 Classes of Instructions

The e600 core instructions belong to one of the following three classes:

- Defined
- Illegal
- Reserved

Note that while the definitions of these terms are consistent among the processors that implement the PowerPC architecture, the assignment of these classifications is not. For example, PowerPC instructions defined for 64-bit implementations are treated as illegal by 32-bit implementations such as the e600 core.

The class is determined by examining the primary opcode and the extended opcode, if any. If the opcode, or combination of opcode and extended opcode, is not that of a defined instruction or of a reserved instruction, the instruction is illegal.

Instruction encodings that are now illegal can become assigned to instructions in the architecture or can be reserved by being assigned to processor-specific instructions.

6.3.1.1 Definition of Boundedly Undefined

If instructions are encoded with incorrectly set bits in reserved fields, the results on execution can be said to be boundedly undefined. If a user-level program executes the incorrectly coded instruction, the resulting undefined results are bounded in that a spurious change from user to supervisor state is not allowed, and

the level of privilege exercised by the program in relation to memory access and other system resources cannot be exceeded. Boundedly undefined results for a given instruction can vary between implementations and between execution attempts in the same implementation.

6.3.1.2 Defined Instruction Class

Defined instructions are guaranteed to be supported in all implementations of the PowerPC architecture, except as stated in the instruction descriptions in Chapter 8, "Instruction Set," of the *Programming Environments Manual*. The e600 core provides hardware support for all instructions defined for 32-bit implementations. It does not support the optional **fsqrt**, **fsqrts**, and **tlbia** instructions.

A processor invokes the illegal instruction error handler (part of the program interrupt) when it encounters a PowerPC instruction that has not been implemented. The instruction can be emulated in software, as required.

A defined instruction can have invalid forms. The e600 core provides limited support for instructions represented in an invalid form.

6.3.1.3 Illegal Instruction Class

Illegal instructions can be grouped into the following categories:

- Instructions not defined in the PowerPC architecture.The following primary opcodes are defined as illegal, but can be used in future extensions to the architecture:
  1, 5, 6, 9, 22, 56, 57, 60, 61

  Future versions of the PowerPC architecture can define any of these instructions to perform new functions.
- Instructions defined in the PowerPC architecture but not implemented in a specific implementation. For example, instructions that can be executed on 64-bit processors that implement the PowerPC architecture are considered illegal by 32-bit processors such as the e600 core.

  The following primary opcodes are defined for 64-bit implementations only and are illegal on the e600 core:

  2, 30, 58, 62
- All unused extended opcodes are illegal. The unused extended opcodes can be determined from information in the "e600 Core Instruction Set Listings" appendix of the *e600 PowerPC Core Reference Manual* and Section 6.3.1.4, "Reserved Instruction Class." Notice that extended opcodes for instructions defined only for 64-bit implementations are illegal in 32-bit implementations, and vice versa. The following primary opcodes have unused extended opcodes:

  17, 19, 31, 59, 63 (Primary opcodes 30 and 62 are illegal for all 32-bit implementations, but as 64-bit opcodes, they have some unused extended opcodes.)
- An instruction consisting of only zeros is guaranteed to be an illegal instruction. This increases the probability that an attempt to execute data or memory that was not initialized invokes the system illegal instruction error handler (a program interrupt). Note that if only the primary opcode consists of all zeros, the instruction is considered a reserved instruction, as described in Section 6.3.1.4, "Reserved Instruction Class."

The e600 core invokes the system illegal instruction error handler (a program interrupt) when it detects any instruction from this class or any instructions defined only for 64-bit implementations.

See the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual* for additional information about illegal and invalid instruction interrupts. Except for an instruction consisting of binary zeros, illegal instructions are available for additions to the PowerPC architecture.

6.3.1.4 Reserved Instruction Class

Reserved instructions are allocated to specific implementation-dependent purposes not defined by the PowerPC architecture. Attempting to execute a reserved instruction that has not been implemented invokes the illegal instruction error handler (a program interrupt). See “Program Interrupt (0x0_0700),” in Chapter 6, “Interrupts,” in the *Programming Environments Manual* for information about illegal and invalid instruction interrupts.

The PowerPC architecture defines four types of reserved instructions:

- Instructions in the POWER architecture not part of the PowerPC UISA. For details on POWER architecture incompatibilities and how they are handled by processors that implement the PowerPC architecture, see Appendix B, “POWER Architecture Cross Reference,” in the *Programming Environments Manual*.
- Implementation-specific instructions required for the processor to conform to the PowerPC architecture (none of these are implemented in the e600 core)
- All other implementation-specific instructions
- Architecturally allowed extended opcodes

6.3.2 Addressing Modes

This section provides an overview of conventions for addressing memory and for calculating effective addresses as defined by the PowerPC architecture for 32-bit implementations. For more detailed information, see “Conventions,” in Chapter 4, “Addressing Modes and Instruction Set Summary,” of the *Programming Environments Manual*.

6.3.2.1 Memory Addressing

A program references memory using the effective (logical) address computed by the processor when it executes a memory access or branch instruction or when it fetches the next sequential instruction.

Bytes in memory are numbered consecutively starting with zero. Each number is the address of the corresponding byte.

6.3.2.2 Memory Operands

Memory operands can be bytes, half-words, words, double-words, quad-words or, for the load/store multiple and load/store string instructions, a sequence of bytes or words. The address of a memory operand is the address of its first byte (that is, of its lowest-numbered byte). Operand length is implicit for each instruction. The PowerPC architecture supports both big-endian and little-endian byte ordering. The

default byte and bit ordering is big endian. See “Byte Ordering,” in Chapter 3, “Operand Conventions,” of the *Programming Environments Manual* for more information about big- and little-endian byte ordering.

The operand of a single-register memory access instruction has a natural alignment boundary equal to the operand length; that is, the natural address of an operand is an integral multiple of its length. A memory operand is said to be aligned if it is aligned at its natural boundary; otherwise it is misaligned. For a detailed discussion about memory operands, see Chapter 3, “Operand Conventions,” of the *Programming Environments Manual*.

6.3.2.3 Effective Address Calculation

An effective address is the 32-bit sum computed by the processor when executing a memory access or branch instruction or when fetching the next sequential instruction. For a memory access instruction, if the sum of the effective address and the operand length exceeds the maximum effective address, the memory operand is considered to wrap around from the maximum effective address through effective address 0, as described in the following paragraphs.

Effective address computations for both data and instruction accesses use 32-bit unsigned binary arithmetic. A carry from bit 0 is ignored.

Load and store operations have the following modes of effective address generation:

- EA = (**r**A|0) + offset (including offset = 0) (register indirect with immediate index)
- EA = (**r**A|0) + **r**B (register indirect with index)

Refer to Section 6.3.4.3.2, “Integer Load and Store Address Generation,” for a detailed description of effective address generation for load and store operations.

Branch instructions have three categories of effective address generation:

- Immediate
- Link register indirect
- Count register indirect

6.3.2.4 Synchronization

The synchronization described in this section refers to the state of the processor that is performing the synchronization.

6.3.2.4.1 Context Synchronization

The System Call (**sc**) and Return from Interrupt (**rfi**) instructions perform context synchronization by allowing previously issued instructions to complete before performing a change in context. Execution of one of these instructions ensures the following:

- No higher priority interrupt exists (**sc**).
- All previous instructions have completed to a point where they can no longer cause an interrupt. If a prior memory access instruction causes direct-store error interrupts, the results are guaranteed to be determined before this instruction is executed.

- Previous instructions complete execution in the context (privilege, protection, and address translation) under which they were issued.
- The instructions following the **sc** or **rfi** instruction execute in the context established by these instructions.

Modifying certain registers requires software synchronization to follow certain register dependencies. Table 6-37 defines specific synchronization procedures that are required when using various SPRs and specific bits within SPRs. Context synchronizing instructions that can be used are: **isync**, **sc**, **rfi**, and any interrupt other than system reset and machine check. If multiple bits are being modified that have different synchronization requirements, the most restrictive requirements can be used. However, a **mtspr** instruction to modify either HID0[ICE] or HID0[ICFI] should not also modify other HID0 bits that requires synchronization.

Table 6-37. Control Registers Synchronization Requirements

Register	Bits	Synchronization Requirements
BAMR	Any	A context synchronizing instruction must follow the **mtspr**.
DABR	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
DBATs	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
EAR	Any	A **dssall** and **sync** must precede the **mtspr** and then a **sync** and a context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing register.

Table 6-37. Control Registers Synchronization Requirements (continued)

Register	Bits	Synchronization Requirements
HID0	BHTCLR	A context synchronizing instruction must precede an **mtspr** and a branch instruction should follow. The branch instruction may be either conditional or unconditional. It ensures that all subsequent branch instructions see the newly initialized BHT values. For correct results, the BHT should be disabled (HID0[BHT] = 0) before setting BHTCLR.
	BHT	A context synchronizing instruction must follow the **mtspr**.
	BTIC	
	DPM	
	FOLD	
	LRSTK	
	NAP	
	NHR	
	SLEEP	
	SPD	
	TBEN	
	DCE	A **dssall** and **sync** must precede an **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the HID0[DCE] or HID0[DCFI] bit.
	DCFI	
	DLOCK	
	NOPDST	
	STEN	
	ICE	A context synchronizing instruction must immediately follow an **mtspr**. An **mtspr** instruction for HID0 should not modify either of these bits at the same time it modifies another bit that requires additional synchronization.
	ICFI	
	ILOCK	A context synchronizing instruction must precede and follow an **mtspr**.
	NOPTI	An **mtspr** must follow a **sync** and a context synchronizing instruction.
	SGE	
	XAEN	A **dssall** and **sync** must precede an **mtspr** and then a **sync** and a context-synchronizing instruction must follow. Alteration of HID0[XAEN] must be done with caches and translation disabled. The caches and TLBs must be flushed before they are re-enabled after the XAEN bit is altered. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the HID0[XAEN] bit.
HID1	Any	A **sync** and context synchronizing instruction must follow an **mtspr**.
IABR	Any	A context synchronizing instruction must follow an **mtspr**.
IBATs	Any	A context synchronizing instruction must follow an **mtspr**.

Table 6-37. Control Registers Synchronization Requirements (continued)

Register	Bits	Synchronization Requirements
ICTRL	EDCE	A **dssall** and **sync** must precede an **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the ICTRL[EDCE] bit.
	ICWL	A context synchronizing instruction must precede and follow an **mtspr**.
	EICE	
L2ERRDIS	Any	A **sync** must precede an **mtspr** and then a **sync** and **isync** must follow.
LDSTCR	Any	A **dssall** and **sync** must precede an **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
MSR	BE	A context synchronizing instruction must follow an **mtmsr** instruction.
	VEC	
	FE0	
	FE1	
	FP	
	SE	
	IR	A context synchronizing instruction must follow an **mtmsr**. When changing the MSR[IR] bit the context synchronizing instruction must reside at both the untranslated and the translated address following the **mtmsr**.
	DR	A **dssall** and **sync** must precede an **mtmsr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the MSR[DR] or MSR[PR] bit.
	PR	
	LE	A **dssall** and **sync** must precede an **rfi** to guarantee a solid context boundary. Note that if a user is not using AltiVec data streaming instructions, a **dssall** is not necessary before accessing MSR[LE].
	POW	A **dssall** and **sync** must precede an **mtmsr** instruction and then a context synchronizing instruction must follow.
MSSCR0	Any	A **dssall** and **sync** must precede an **mtspr** instruction and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
SDR1	Any	A **dssall** and **sync** must precede an **mtspr** and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
SR0–SR15	Any	A **dssall** and **sync** must precede an **mtsr** or **mtsrin** instruction and then a **sync** and context synchronizing instruction must follow. Note that if a user is not using the AltiVec data streaming instructions, then a **dssall** is not necessary prior to accessing the register.
Other registers or bits	—	No special synchronization requirements.

6.3.2.4.2 Execution Synchronization

An instruction is execution synchronizing if all previously initiated instructions appear to have completed before the instruction is initiated or, in the case of **sync** and **isync**, before the instruction completes. For example, the Move to Machine State Register (**mtmsr**) instruction is execution synchronizing. It ensures that all preceding instructions have completed execution and cannot cause an interrupt before the instruction executes, but does not ensure subsequent instructions execute in the newly established environment. For example, if the **mtmsr** sets the MSR[PR] bit, unless an **isync** immediately follows the **mtmsr** instruction, a privileged instruction could be executed or privileged access could be performed without causing an interrupt even though MSR[PR] indicates user mode.

6.3.2.4.3 Instruction-Related Interrupts

There are two kinds of interrupts in the e600 core—those caused directly by the execution of an instruction and those caused by an asynchronous event. Either can cause components of the system software to be invoked.

Interrupts can be caused directly by the execution of an instruction as follows:

- An attempt to execute an illegal instruction causes the illegal instruction (program interrupt) handler to be invoked. An attempt by a user-level program to execute the supervisor-level instructions listed below causes the privileged instruction (program interrupt) handler to be invoked. The e600 core provides the following supervisor-level instructions—**dcbi**, **mfmsr**, **mfspr**, **mfsr**, **mfsrin**, **mtmsr**, **mtspr**, **mtsr**, **mtsrin**, **rfi**, **tlbie**, and **tlbsync**. Note that the privilege level of the **mfspr** and **mtspr** instructions depends on the SPR encoding.
- Any **mtspr**, **mfspr**, or **mftb** instruction with an invalid SPR (or TBR) field causes an illegal type program interrupt. Likewise, a program interrupt is taken if user-level software tries to access a supervisor-level SPR. An **mtspr** instruction executing in supervisor mode (MSR[PR] = 0) with the SPR field specifying PVR (read-only register) executes as a no-op.
- An attempt to access memory that is not available (page fault) causes the ISI or DSI interrupt handler to be invoked.
- The execution of an **sc** instruction invokes the system call interrupt handler that permits a program to request the system to perform a service.
- The execution of a trap instruction invokes the program interrupt trap handler.
- The execution of an instruction that causes a floating-point exception while exceptions are enabled in the MSR invokes the program interrupt handler.

A detailed description of exception conditions is provided in the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual.*

6.3.3 Instruction Set Overview

This section provides a brief overview of the PowerPC instructions implemented in the e600 core and highlights any special information with respect to how the e600 core implements a particular instruction. Note that the categories used in this section correspond to those used in Chapter 4, “Addressing Modes and Instruction Set Summary,” in the *Programming Environments Manual*. These categorizations are somewhat arbitrary, are provided for the convenience of the programmer, and do not necessarily reflect the PowerPC architecture specification.

Note that some instructions have the following optional features:

- CR Update—The dot (**.**) suffix on the mnemonic enables the update of the CR.
- Overflow option—The **o** suffix indicates that the overflow bit in the XER is enabled.

6.3.4 PowerPC UISA Instructions

The PowerPC UISA includes the base user-level instruction set (excluding a few user-level cache control, synchronization, and time base instructions), user-level registers, programming model, data types, and addressing modes. This section discusses the instructions defined in the UISA.

6.3.4.1 Integer Instructions

This section describes the integer instructions. These consist of the following:

- Integer arithmetic instructions
- Integer compare instructions
- Integer logical instructions
- Integer rotate and shift instructions

Integer instructions use the content of the GPRs as source operands and place results into GPRs, the XER register, and condition register (CR) fields.

6.3.4.1.1 Integer Arithmetic Instructions

Table 6-38 lists the integer arithmetic instructions for the processors that implement the PowerPC architecture.

Table 6-38. Integer Arithmetic Instructions

Name	Mnemonic	Syntax
Add Immediate	**addi**	**r**D,**r**A,SIMM
Add Immediate Shifted	**addis**	**r**D,**r**A,SIMM
Add	**add** (**add. addo addo.**)	**r**D,**r**A,**r**B
Subtract From	**subf** (**subf. subfo subfo.**)	**r**D,**r**A,**r**B
Add Immediate Carrying	**addic**	**r**D,**r**A,SIMM
Add Immediate Carrying and Record	**addic.**	**r**D,**r**A,SIMM
Subtract from Immediate Carrying	**subfic**	**r**D,**r**A,SIMM
Add Carrying	**addc** (**addc. addco addco.**)	**r**D,**r**A,**r**B
Subtract from Carrying	**subfc** (**subfc. subfco subfco.**)	**r**D,**r**A,**r**B
Add Extended	**adde** (**adde. addeo addeo.**)	**r**D,**r**A,**r**B
Subtract from Extended	**subfe** (**subfe. subfeo subfeo.**)	**r**D,**r**A,**r**B
Add to Minus One Extended	**addme** (**addme. addmeo addmeo.**)	**r**D,**r**A
Subtract from Minus One Extended	**subfme** (**subfme. subfmeo subfmeo.**)	**r**D,**r**A

Table 6-38. Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Add to Zero Extended	**addze** (**addze. addzeo addzeo.**)	**r**D,**r**A
Subtract from Zero Extended	**subfze** (**subfze. subfzeo subfzeo.**)	**r**D,**r**A
Negate	**neg** (**neg. nego nego.**)	**r**D,**r**A
Multiply Low Immediate	**mulli**	**r**D,**r**A,SIMM
Multiply Low Word	**mullw** (**mullw. mullwo mullwo.**)	**r**D,**r**A,**r**B
Multiply High Word	**mulhw** (**mulhw.**)	**r**D,**r**A,**r**B
Multiply High Word Unsigned	**mulhwu** (**mulhwu.**)	**r**D,**r**A,**r**B
Divide Word	**divw** (**divw. divwo divwo.**)	**r**D,**r**A,**r**B
Divide Word Unsigned	**divwu divwu. divwuo divwuo.**	**r**D,**r**A,**r**B

Although there is no Subtract Immediate instruction, its effect can be achieved by using an **addi** instruction with the immediate operand negated. Simplified mnemonics are provided that include this negation. The **subf** instructions subtract the second operand (**r**A) from the third operand (**r**B). Simplified mnemonics are provided in which the third operand is subtracted from the second operand. See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for examples.

The UISA states that an implementation that executes instructions that set the overflow enable bit (OE) or the carry bit (CA) can either execute these instructions slowly or prevent execution of the subsequent instruction until the operation completes. The “Instruction Timing” chapter of the *e600 PowerPC Core Reference Manual* describes how the e600 core handles CR dependencies. The summary overflow bit (SO) and overflow bit (OV) in the XER register are set to reflect an overflow condition of a 32-bit result. This can happen only when OE = 1.

6.3.4.1.2 Integer Compare Instructions

The integer compare instructions algebraically or logically compare the contents of register **r**A with either the zero-extended value of the UIMM operand, the sign-extended value of the SIMM operand, or the contents of **r**B. The comparison is signed for the **cmpi** and **cmp** instructions, and unsigned for the **cmpli** and **cmpl** instructions. Table 6-39 summarizes the integer compare instructions.

Table 6-39. Integer Compare Instructions

Name	Mnemonic	Syntax
Compare Immediate	**cmpi**	**crf**D,L,**r**A,SIMM
Compare	**cmp**	**crf**D,L,**r**A,**r**B
Compare Logical Immediate	**cmpli**	**crf**D,L,**r**A,UIMM
Compare Logical	**cmpl**	**crf**D,L,**r**A,**r**B

The **crf**D operand can be omitted if the result of the comparison is to be placed in CR0. Otherwise the target CR field must be specified in **crf**D, using an explicit field number.

For information on simplified mnemonics for the integer compare instructions see Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual*.

6.3.4.1.3 Integer Logical Instructions

The logical instructions shown in Table 6-40 perform bit-parallel operations on the specified operands. Logical instructions with the CR updating enabled (uses dot suffix) and instructions **andi.** and **andis.** set CR field CR0 to characterize the result of the logical operation. Logical instructions do not affect XER[SO], XER[OV], or XER[CA].

See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for simplified mnemonic examples for integer logical operations.

Table 6-40. Integer Logical Instructions

Name	Mnemonic	Syntax	Implementation Notes
AND Immediate	**andi.**	**r**A,**r**S,UIMM	—
AND Immediate Shifted	**andis.**	**r**A,**r**S,UIMM	—
OR Immediate	**ori**	**r**A,**r**S,UIMM	The PowerPC architecture defines **ori r**0,**r**0,0 as the preferred form for the no-op instruction. The dispatcher discards this instruction and only dispatches it to the completion queue, but not to any execution unit.
OR Immediate Shifted	**oris**	**r**A,**r**S,UIMM	—
XOR Immediate	**xori**	**r**A,**r**S,UIMM	—
XOR Immediate Shifted	**xoris**	**r**A,**r**S,UIMM	—
AND	**and** (**and.**)	**r**A,**r**S,**r**B	—
OR	**or (or.)**	**r**A,**r**S,**r**B	—
XOR	**xor (xor.)**	**r**A,**r**S,**r**B	—
NAND	**nand (nand.)**	**r**A,**r**S,**r**B	—
NOR	**nor (nor.)**	**r**A,**r**S,**r**B	—
Equivalent	**eqv (eqv.)**	**r**A,**r**S,**r**B	—
AND with Complement	**andc (andc.)**	**r**A,**r**S,**r**B	—
OR with Complement	**orc (orc.)**	**r**A,**r**S,**r**B	—
Extend Sign Byte	**extsb (extsb.)**	**r**A,**r**S	—
Extend Sign Half Word	**extsh (extsh.)**	**r**A,**r**S	—
Count Leading Zeros Word	**cntlzw (cntlzw.)**	**r**A,**r**S	—

6.3.4.1.4 Integer Rotate and Shift Instructions

Rotation operations are performed on data from a GPR, and the result, or a portion of the result, is returned to a GPR. See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for a complete list of simplified mnemonics that allows simpler coding of often-used functions such as clearing

the leftmost or rightmost bits of a register, left-justifying or right-justifying an arbitrary field, and simple rotates and shifts.

Integer rotate instructions rotate the contents of a register. The result of the rotation is either inserted into the target register under control of a mask (if a mask bit is 1 the associated bit of the rotated data is placed into the target register, and if the mask bit is 0 the associated bit in the target register is unchanged), or ANDed with a mask before being placed into the target register.

The integer rotate instructions are summarized in Table 6-41.

Table 6-41. Integer Rotate Instructions

Name	Mnemonic	Syntax
Rotate Left Word Immediate then AND with Mask	**rlwinm** (**rlwinm.**)	**r**A,**r**S,SH,MB,ME
Rotate Left Word then AND with Mask	**rlwnm** (**rlwnm.**)	**r**A,**r**S,**r**B,MB,ME
Rotate Left Word Immediate then Mask Insert	**rlwimi** (**rlwimi.**)	**r**A,**r**S,SH,MB,ME

The integer shift instructions perform left and right shifts. Immediate-form logical (unsigned) shift operations are obtained by specifying masks and shift values for certain rotate instructions. Simplified mnemonics (shown in Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual*) are provided to make coding of such shifts simpler and easier to understand.

Multiple-precision shifts can be programmed as shown in Appendix C, “Multiple-Precision Shifts,” in the *Programming Environments Manual*. The integer shift instructions are summarized in Table 6-42.

Table 6-42. Integer Shift Instructions

Name	Mnemonic	Syntax
Shift Left Word	**slw** (**slw.**)	**r**A,**r**S,**r**B
Shift Right Word	**srw** (**srw.**)	**r**A,**r**S,**r**B
Shift Right Algebraic Word Immediate	**srawi** (**srawi.**)	**r**A,**r**S,SH
Shift Right Algebraic Word	**sraw** (**sraw.**)	**r**A,**r**S,**r**B

6.3.4.2 Floating-Point Instructions

This section describes the floating-point instructions, which include the following:

- Floating-point arithmetic instructions
- Floating-point multiply-add instructions
- Floating-point rounding and conversion instructions
- Floating-point compare instructions
- Floating-point status and control register instructions
- Floating-point move instructions

See Section 6.3.4.3, “Load and Store Instructions,” for information about floating-point loads and stores.

The PowerPC architecture supports a floating-point system as defined in IEEE Std. 754, but requires software support to conform with that standard. All floating-point operations conform to IEEE Std.754, except if software sets the non-IEEE mode bit (FPSCR[NI]).

6.3.4.2.1 Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions are summarized in Table 6-43.

Table 6-43. Floating-Point Arithmetic Instructions

Name	Mnemonic	Syntax
Floating Add (Double-Precision)	**fadd (fadd.)**	**fr**D,**fr**A,**fr**B
Floating Add Single	**fadds (fadds.)**	**fr**D,**fr**A,**fr**B
Floating Subtract (Double-Precision)	**fsub (fsub.)**	**fr**D,**fr**A,**fr**B
Floating Subtract Single	**fsubs (fsubs.)**	**fr**D,**fr**A,**fr**B
Floating Multiply (Double-Precision)	**fmul (fmul.)**	**fr**D,**fr**A,**fr**C
Floating Multiply Single	**fmuls (fmuls.)**	**fr**D,**fr**A,**fr**C
Floating Divide (Double-Precision)	**fdiv (fdiv.)**	**fr**D,**fr**A,**fr**B
Floating Divide Single	**fdivs (fdivs.)**	**fr**D,**fr**A,**fr**B
Floating Reciprocal Estimate Single[1]	**fres (fres.)**	**fr**D,**fr**B
Floating Reciprocal Square Root Estimate[1]	**frsqrte (frsqrte.)**	**fr**D,**fr**B
Floating Select[1]	**fsel**	**fr**D,**fr**A,**fr**C,**fr**B

[1] These instructions are optional in the PowerPC architecture.

All single-precision arithmetic instructions are performed using a double-precision format. The floating-point architecture is a single-pass implementation for double-precision products. In most cases, a single-precision instruction using only single-precision operands, in double-precision format, has the same latency as its double-precision equivalent.

6.3.4.2.2 Floating-Point Multiply-Add Instructions

These instructions combine multiply and add operations without an intermediate rounding operation. The floating-point multiply-add instructions are summarized in Table 6-44.

Table 6-44. Floating-Point Multiply-Add Instructions

Name	Mnemonic	Syntax
Floating Multiply-Add (Double-Precision)	**fmadd (fmadd.)**	**fr**D,**fr**A,**fr**C,**fr**B
Floating Multiply-Add Single	**fmadds (fmadds.)**	**fr**D,**fr**A,**fr**C,**fr**B
Floating Multiply-Subtract (Double-Precision)	**fmsub (fmsub.)**	**fr**D,**fr**A,**fr**C,**fr**B
Floating Multiply-Subtract Single	**fmsubs (fmsubs.)**	**fr**D,**fr**A,**fr**C,**fr**B
Floating Negative Multiply-Add (Double-Precision)	**fnmadd (fnmadd.)**	**fr**D,**fr**A,**fr**C,**fr**B
Floating Negative Multiply-Add Single	**fnmadds (fnmadds.)**	**fr**D,**fr**A,**fr**C,**fr**B

Table 6-44. Floating-Point Multiply-Add Instructions (continued)

Name	Mnemonic	Syntax
Floating Negative Multiply-Subtract (Double-Precision)	**fnmsub (fnmsub.)**	**fr**D,**fr**A,**fr**C,**fr**B
Floating Negative Multiply-Subtract Single	**fnmsubs (fnmsubs.)**	**fr**D,**fr**A,**fr**C,**fr**B

6.3.4.2.3 Floating-Point Rounding and Conversion Instructions

The Floating Round to Single-Precision (**frsp**) instruction is used to truncate a 64-bit double-precision number to a 32-bit single-precision floating-point number. The floating-point convert instructions convert a 64-bit double-precision floating-point number to a 32-bit signed integer number.

Examples of uses of these instructions to perform various conversions can be found in Appendix D, “Floating-Point Models,” in the *Programming Environments Manual*.

Table 6-45. Floating-Point Rounding and Conversion Instructions

Name	Mnemonic	Syntax
Floating Round to Single	**frsp (frsp.)**	**fr**D,**fr**B
Floating Convert to Integer Word	**fctiw (fctiw.)**	**fr**D,**fr**B
Floating Convert to Integer Word with Round toward Zero	**fctiwz (fctiwz.)**	**fr**D,**fr**B

6.3.4.2.4 Floating-Point Compare Instructions

Floating-point compare instructions compare the contents of two floating-point registers. The comparison ignores the sign of zero (that is +0 = –0). The floating-point compare instructions are summarized in Table 6-46.

Table 6-46. Floating-Point Compare Instructions

Name	Mnemonic	Syntax
Floating Compare Unordered	**fcmpu**	**crf**D,**fr**A,**fr**B
Floating Compare Ordered	**fcmpo**	**crf**D,**fr**A,**fr**B

6.3.4.2.5 Floating-Point Status and Control Register Instructions

Every FPSCR instruction appears to synchronize the effects of all floating-point instructions executed by a given processor. Executing an FPSCR instruction ensures that all floating-point instructions previously initiated by the given processor appear to have completed before the FPSCR instruction is initiated and that no subsequent floating-point instructions appear to be initiated by the given processor until the FPSCR instruction has completed. The FPSCR instructions are summarized in Table 6-47.

Table 6-47. Floating-Point Status and Control Register Instructions

Name	Mnemonic	Syntax
Move from FPSCR	**mffs (mffs.)**	**fr**D
Move to Condition Register from FPSCR	**mcrfs**	**crf**D,**crf**S

Table 6-47. Floating-Point Status and Control Register Instructions (continued)

Name	Mnemonic	Syntax
Move to FPSCR Field Immediate	**mtfsfi (mtfsfi.)**	**crf**D,IMM
Move to FPSCR Fields	**mtfsf (mtfsf.)**	FM,**fr**B
Move to FPSCR Bit 0	**mtfsb0 (mtfsb0.)**	**crb**D
Move to FPSCR Bit 1	**mtfsb1 (mtfsb1.)**	**crb**D

Implementation Note—The PowerPC architecture states that in some implementations, the Move to FPSCR Fields (**mtfsf**) instruction can perform more slowly when only some of the fields are updated as opposed to all of the fields. In the e600 core, there is no degradation of performance.

6.3.4.2.6 Floating-Point Move Instructions

Floating-point move instructions copy data from one FPR to another. The floating-point move instructions do not modify the FPSCR. The CR update option in these instructions controls the placing of result status into CR1. Table 6-48 summarizes the floating-point move instructions.

Table 6-48. Floating-Point Move Instructions

Name	Mnemonic	Syntax
Floating Move Register	**fmr (fmr.)**	**fr**D,**fr**B
Floating Negate	**fneg (fneg.)**	**fr**D,**fr**B
Floating Absolute Value	**fabs (fabs.)**	**fr**D,**fr**B
Floating Negative Absolute Value	**fnabs (fnabs.)**	**fr**D,**fr**B

6.3.4.3 Load and Store Instructions

Load and store instructions are issued and translated in program order; however, the accesses can occur out of order. Synchronizing instructions are provided to enforce strict ordering. This section describes the load and store instructions, which consist of the following:

- Integer load instructions
- Integer store instructions
- Integer load and store with byte-reverse instructions
- Integer load and store multiple instructions
- Floating-point load instructions
- Floating-point store instructions
- Memory synchronization instructions

Implementation Note—The following describes how the e600 core handles misalignment:

The e600 core provides hardware support for misaligned memory accesses. It performs those accesses within a single cycle if the operand lies within a double-word boundary. Misaligned memory accesses that cross a double-word boundary degrade performance.

Although many misaligned memory accesses are supported in hardware, the frequent use of them is discouraged because they can compromise the overall performance of the processor. Only one outstanding misalignment at a time is supported which means it is non-pipelined.

Accesses that cross a translation boundary can be restarted. That is, a misaligned access that crosses a page boundary is completely restarted if the second portion of the access causes a page fault. This can cause the first access to be repeated.

On some processors, such as the MPC603e, a TLB reload operation causes an instruction restart. On the e600 core, TLB reloads are performed transparently (if hardware table search operations are enabled—HID0[STEN] = 0) and only a page fault causes a restart. If software table searching is enabled (HID0[STEN] = 1) on the e600 core, a TLB miss causes an instruction restart (as it causes a TLB miss interrupt)

6.3.4.3.1 Self-Modifying Code

When a processor modifies a memory location that can be contained in the instruction cache, software must ensure that memory updates are visible to the instruction fetching mechanism. This can be achieved by executing the following instruction sequence (using either **dcbst** or **dcbf**):

```
dcbst (or dcbf)|update memory
sync     |wait for update
icbi     |remove (invalidate) copy in instruction cache
sync     |ensure that ICBI invalidate at the icache has completed
isync    |remove copy in own instruction buffer
```

These operations are required because the data cache is a write-back cache. Because instruction fetching bypasses the data cache, changes to items in the data cache cannot be reflected in memory until the fetch operations complete. The **sync** after the **icbi** is required to ensure that the **icbi** invalidation has completed in the instruction cache.

Special care must be taken to avoid coherency paradoxes in systems that implement unified secondary caches (like the e600 core), and designers should carefully follow the guidelines for maintaining cache coherency that are provided in the VEA, and discussed in Chapter 5, “Cache Model and Memory Coherency,” in the *Programming Environments Manual*.

6.3.4.3.2 Integer Load and Store Address Generation

Integer load and store operations generate effective addresses using register indirect with immediate index mode, register indirect with index mode, or register indirect mode. See Section 6.3.2.3, “Effective Address Calculation,” for information about calculating effective addresses. Note that in some implementations, operations that are not naturally aligned can suffer performance degradation. Refer to the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual* for additional information about load and store address alignment interrupts.

6.3.4.3.3 Register Indirect Integer Load Instructions

For integer load instructions, the byte, half-word, word, or double-word addressed by the EA (effective address) is loaded into **r**D. Many integer load instructions have an update form, in which **r**A is updated with the generated effective address. For these forms, if **r**A ≠ 0 and **r**A ≠ **r**D (otherwise invalid), the EA

is placed into **r**A and the memory element (byte, half-word, word, or double-word) addressed by the EA is loaded into **r**D. Note that the PowerPC architecture defines load with update instructions with operand **r**A = 0 or **r**A = **r**D as invalid forms.

Implementation Notes—The following notes describe the e600 core implementation of integer load instructions:

- The PowerPC architecture cautions programmers that some implementations of the architecture can execute the load half algebraic (**lha**, **lhax**) instructions with greater latency than other types of load instructions. This is not the case for the e600 core; these instructions operate with the same latency as other load instructions.
- The PowerPC architecture cautions programmers that some implementations of the architecture can run the load/store byte-reverse (**lhbrx**, **lbrx**, **sthbrx**, **stwbrx**) instructions with greater latency than other types of load/store instructions. This is not the case for the e600 core. These instructions operate with the same latency as the other load/store instructions.
- The PowerPC architecture describes some preferred instruction forms for load and store multiple instructions and integer move assist instructions that can perform better than other forms in some implementations. None of these preferred forms affect instruction performance on the e600 core. Usage of load/store string instruction is discouraged.
- The PowerPC architecture defines the **lwarx** and **stwcx.** as a way to update memory atomically. In the e600 core, reservations are made on behalf of aligned 32-byte sections of the memory address space. Executing **lwarx** and **stwcx.** to a page marked write-through does cause a DSI interrupt if the page is marked cacheable write-through (WIM = 10x) or caching-inhibited (WIM = x1x), but as with other memory accesses, DSI interrupts can result for other reasons such as protection violations or page faults.

Table 6-49 summarizes the integer load instructions.

Table 6-49. Integer Load Instructions

Name	Mnemonic	Syntax
Load Byte and Zero	**lbz**	**r**D,**d**(r**A**)
Load Byte and Zero Indexed	**lbzx**	**r**D,**r**A,**r**B
Load Byte and Zero with Update	**lbzu**	**r**D,**d**(r**A**)
Load Byte and Zero with Update Indexed	**lbzux**	**r**D,**r**A,**r**B
Load Half Word and Zero	**lhz**	**r**D,**d**(r**A**)
Load Half Word and Zero Indexed	**lhzx**	**r**D,**r**A,**r**B
Load Half Word and Zero with Update	**lhzu**	**r**D,**d**(r**A**)
Load Half Word and Zero with Update Indexed	**lhzux**	**r**D,**r**A,**r**B
Load Half Word Algebraic	**lha**	**r**D,**d**(r**A**)
Load Half Word Algebraic Indexed	**lhax**	**r**D,**r**A,**r**B
Load Half Word Algebraic with Update	**lhau**	**r**D,**d**(r**A**)
Load Half Word Algebraic with Update Indexed	**lhaux**	**r**D,**r**A,**r**B

Table 6-49. Integer Load Instructions (continued)

Name	Mnemonic	Syntax
Load Word and Zero	**lwz**	**r**D,**d**(**r**A)
Load Word and Zero Indexed	**lwzx**	**r**D,**r**A,**r**B
Load Word and Zero with Update	**lwzu**	**r**D,**d**(**r**A)
Load Word and Zero with Update Indexed	**lwzux**	**r**D,**r**A,**r**B

6.3.4.3.4 Integer Store Instructions

For integer store instructions, the contents of **r**S are stored into the byte, half-word, word or double-word in memory addressed by the EA (effective address). Many store instructions have an update form, in which **r**A is updated with the EA. For these forms, the following rules apply:

- If **r**A ≠ 0, the effective address is placed into **r**A.
- If **r**S = **r**A, the contents of register **r**S are copied to the target memory element, then the generated EA is placed into **r**A (**r**S).

The PowerPC architecture defines store with update instructions with **r**A = 0 as an invalid form. In addition, it defines integer store instructions with the CR update option enabled (Rc field, bit 31, in the instruction encoding = 1) to be an invalid form. Table 6-50 summarizes the integer store instructions.

Table 6-50. Integer Store Instructions

Name	Mnemonic	Syntax
Store Byte	**stb**	**r**S,**d**(**r**A)
Store Byte Indexed	**stbx**	**r**S,**r**A,**r**B
Store Byte with Update	**stbu**	**r**S,**d**(**r**A)
Store Byte with Update Indexed	**stbux**	**r**S,**r**A,**r**B
Store Half Word	**sth**	**r**S,**d**(**r**A)
Store Half Word Indexed	**sthx**	**r**S,**r**A,**r**B
Store Half Word with Update	**sthu**	**r**S,**d**(**r**A)
Store Half Word with Update Indexed	**sthux**	**r**S,**r**A,**r**B
Store Word	**stw**	**r**S,**d**(**r**A)
Store Word Indexed	**stwx**	**r**S,**r**A,**r**B
Store Word with Update	**stwu**	**r**S,**d**(**r**A)
Store Word with Update Indexed	**stwux**	**r**S,**r**A,**r**B

6.3.4.3.5 Integer Store Gathering

The e600 core performs store gathering for write-through accesses to nonguarded space or to cache-inhibited stores to nonguarded space if the requirements described in the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* are met. These stores are combined in the load/store unit (LSU) to form a double-word or quad-word and are sent out on the MPX bus as a single

operation. However, stores can be gathered only if the successive stores that meet the criteria are queued and pending. The e600 core also performs store merging as described in the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*.

Store gathering takes place regardless of the address order of the stores. The store gathering and merging feature is enabled by setting HID0[SGE].

If store gathering is enabled and the stores do not fall under the above categories, an **eieio** or **sync** instruction must be used to prevent two stores from being gathered.

6.3.4.3.6 Integer Load and Store with Byte-Reverse Instructions

Table 6-51 describes integer load and store with byte-reverse instructions. When used in a system operating with the default big-endian byte order, these instructions have the effect of loading and storing data in little-endian order. Likewise, when used in a system operating with little-endian byte order, these instructions have the effect of loading and storing data in big-endian order. For more information about big-endian and little-endian byte ordering, see “Byte Ordering,” in Chapter 3, “Operand Conventions,” in the *Programming Environments Manual*.

Table 6-51. Integer Load and Store with Byte-Reverse Instructions

Name	Mnemonic	Syntax
Load Half Word Byte-Reverse Indexed	**lhbrx**	**r**D,**r**A,**r**B
Load Word Byte-Reverse Indexed	**lwbrx**	**r**D,**r**A,**r**B
Store Half Word Byte-Reverse Indexed	**sthbrx**	**r**S,**r**A,**r**B
Store Word Byte-Reverse Indexed	**stwbrx**	**r**S,**r**A,**r**B

6.3.4.3.7 Integer Load and Store Multiple Instructions

The load/store multiple instructions are used to move blocks of data to and from the GPRs. The load multiple and store multiple instructions can have operands that require memory accesses crossing a 4-Kbyte page boundary. As a result, these instructions can be interrupted by a DSI interrupt associated with the address translation of the second page.

The PowerPC architecture defines the Load Multiple Word (**lmw**) instruction with **r**A in the range of registers to be loaded as an invalid form.

Table 6-52. Integer Load and Store Multiple Instructions

Name	Mnemonic	Syntax
Load Multiple Word	**lmw**	**r**D,**d(r**A**)**
Store Multiple Word	**stmw**	**r**S,**d(r**A**)**

6.3.4.3.8 Integer Load and Store String Instructions

The integer load and store string instructions allow movement of data from memory to registers or from registers to memory without concern for alignment. These instructions can be used for a short move between arbitrary memory locations or to initiate a long move between misaligned memory fields.

However, in some implementations, these instructions are likely to have greater latency and take longer to execute, perhaps much longer, than a sequence of individual load or store instructions that produce the same results. Table 6-53 summarizes the integer load and store string instructions.

Table 6-53. Integer Load and Store String Instructions

Name	Mnemonic	Syntax
Load String Word Immediate	**lswi**	**r**D,**r**A,NB
Load String Word Indexed	**lswx**	**r**D,**r**A,**r**B
Store String Word Immediate	**stswi**	**r**S,**r**A,NB
Store String Word Indexed	**stswx**	**r**S,**r**A,**r**B

In the e600 core implementation, operating with little-endian byte order, execution of a load or string instruction will take an alignment interrupt.

Load string and store string instructions can involve operands that are not word-aligned.

For load/store string operations, the e600 core does not combine register values to reduce the number of discrete accesses. However, if store gathering is enabled and the accesses fall under the criteria for store gathering the stores can be combined to enhance performance. At a minimum, additional cache access cycles are required. Usage of load/store string instructions is discouraged.

6.3.4.3.9 Floating-Point Load and Store Address Generation

Floating-point load and store operations generate effective addresses using the register indirect with immediate index addressing mode and register indirect with index addressing mode. Floating-point loads and stores are not supported for direct-store accesses. The use of floating-point loads and stores for direct-store access results in an alignment interrupt.

There are two forms of the floating-point load instruction—single-precision and double-precision operand formats. Because the FPRs support only the floating-point double-precision format, single-precision floating-point load instructions convert single-precision data to double-precision format before loading an operand into an FPR.

Implementation Note—The e600 core treats interrupts as follows:

- The FPU operates in either ignore exceptions mode (MSR[FE0] = MSR[FE1] = 0) or precise mode (any other settings for MSR[FE0,FE1]). For the e600 core, ignore exceptions mode allows floating-point instructions to complete earlier and thus can provide better performance than precise mode.
- The floating-point load and store indexed instructions (**lfsx**, **lfsux**, **lfdx**, **lfdux**, **stfsx**, **stfsux**, **stfdx**, **stfdux**) are invalid when the Rc bit is one.

The PowerPC architecture defines a load with update instruction with **r**A = 0 as an invalid form. Table 6-54 summarizes the floating-point load instructions.

Table 6-54. Floating-Point Load Instructions

Name	Mnemonic	Syntax
Load Floating-Point Single	**lfs**	**fr**D,**d(r**A**)**
Load Floating-Point Single Indexed	**lfsx**	**fr**D,**r**A,**r**B
Load Floating-Point Single with Update	**lfsu**	**fr**D,**d(r**A**)**
Load Floating-Point Single with Update Indexed	**lfsux**	**fr**D,**r**A,**r**B
Load Floating-Point Double	**lfd**	**fr**D,**d(r**A**)**
Load Floating-Point Double Indexed	**lfdx**	**fr**D,**r**A,**r**B
Load Floating-Point Double with Update	**lfdu**	**fr**D,**d(r**A**)**
Load Floating-Point Double with Update Indexed	**lfdux**	**fr**D,**r**A,**r**B

6.3.4.3.10 Floating-Point Store Instructions

This section describes floating-point store instructions. There are three basic forms of the store instruction—single-precision, double-precision, and integer. The integer form is supported by the optional **stfiwx** instruction. Because the FPRs support only floating-point, double-precision format for floating-point data, single-precision floating-point store instructions convert double-precision data to single-precision format before storing the operands. Table 6-55 summarizes the floating-point store instructions.

Table 6-55. Floating-Point Store Instructions

Name	Mnemonic	Syntax
Store Floating-Point Single	**stfs**	**fr**S,**d(r**A**)**
Store Floating-Point Single Indexed	**stfsx**	**fr**S,**r** B
Store Floating-Point Single with Update	**stfsu**	**fr**S,**d(**rA**)**
Store Floating-Point Single with Update Indexed	**stfsux**	**fr**S,**r** B
Store Floating-Point Double	**stfd**	**fr**S,**d(r**A**)**
Store Floating-Point Double Indexed	**stfdx**	**fr**S,**r**B
Store Floating-Point Double with Update	**stfdu**	**fr**S,**d(**rA**)**
Store Floating-Point Double with Update Indexed	**stfdux**	**fr**S,**r** B
Store Floating-Point as Integer Word Indexed[1]	**stfiwx**	**fr**S,**r**B

[1] The **stfiwx** instruction is optional to the PowerPC architecture

Some floating-point store instructions require conversions in the LSU. Table 6-56 shows conversions the LSU makes when executing a Store Floating-Point Single instruction.

Table 6-56. Store Floating-Point Single Behavior

FPR Precision	Data Type	Action
Single	Normalized	Store
Single	Denormalized	Store
Single	Zero, infinity, QNaN	Store
Single	SNaN	Store

Double	Normalized	If (exp ≤ 896)
		then
		Denormalize and Store
		else
		Store

Double	Denormalized	Store zero
Double	Zero, infinity, QNaN	Store
Double	SNaN	Store

Table 6-57 shows the conversions made when performing a Store Floating-Point Double instruction. Most entries in the table indicate that the floating-point value is simply stored. Only in a few cases are any other actions taken.

Table 6-57. Store Floating-Point Double Behavior

FPR Precision	Data Type	Action
Single	Normalized	Store
Single	Denormalized	Normalize and Store
Single	Zero, infinity, QNaN	Store
Single	SNaN	Store
Double	Normalized	Store
Double	Denormalized	Store
Double	Zero, infinity, QNaN	Store
Double	SNaN	Store

Architecturally, all floating-point numbers are represented in double-precision format within the e600 core. Execution of a store floating-point single (**stfs**, **stfsu**, **stfsx**, **stfsux**) instruction requires conversion from double- to single-precision format. If the exponent is not greater than 896, this conversion requires denormalization. The e600 core supports this denormalization by shifting the mantissa one bit at a time. Anywhere from 1 to 23 clock cycles are required to complete the denormalization, depending upon the value to be stored.

Because of how floating-point numbers are implemented in the e600 core, there is also a case when execution of a store floating-point double (**stfd**, **stfdu**, **stfdx**, **stfdux**) instruction can require internal

shifting of the mantissa. This case occurs when the operand of a store floating-point double instruction is a denormalized single-precision value. The value could be the result of a load floating-point single instruction, a single-precision arithmetic instruction, or a floating round to single-precision instruction. In these cases, shifting the mantissa takes from 1 to 23 clock cycles, depending upon the value to be stored. These cycles are incurred during the store.

6.3.4.4 Branch and Flow Control Instructions

Some branch instructions can redirect instruction execution conditionally based on the value of bits in the CR. When the processor encounters one of these instructions, it scans the execution pipelines to determine whether an instruction in progress can affect the particular CR bit. If no interlock is found, the branch can be resolved immediately by checking the bit in the CR and taking the action defined for the branch instruction.

6.3.4.4.1 Branch Instruction Address Calculation

Branch instructions can alter the sequence of instruction execution. Instruction addresses are always assumed to be word aligned; the processors that ignore the two low-order bits of the generated branch target address.

Branch instructions compute the EA of the next instruction address using the following addressing modes:

- Branch relative
- Branch conditional to relative address
- Branch to absolute address
- Branch conditional to absolute address
- Branch conditional to link register
- Branch conditional to count register

Note that in the e600 core, all branch instructions (**b**, **ba**, **bl**, **bla**, **bc**, **bca**, **bcl**, **bcla**, **bclr**, **bclrl**, **bcctr**, **bcctrl**) are executed in the BPU and condition register logical instructions (**crand**, **cror**, **crxor**, **crnand**, **crnor**, **crandc**, **creqv**, **crorc**, and **mcrf**) are executed by the IU2. Some of these instructions can redirect instruction execution conditionally on the value of CR, CTR, or LR bits. When the CR bits resolve, the branch instruction is either marked as correct or mispredicted. Correcting a mispredicted branch requires that the e600 core flush speculatively executed instructions and restore the machine state to immediately after the branch. This correction can be done when all non-speculative instructions older than the mispredicting branch have completed.

6.3.4.4.2 Branch Instructions

Table 6-58 lists the branch instructions provided by the processors that implement the PowerPC architecture. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for the most frequently used forms of branch conditional, compare, trap, rotate and shift, and

certain other instructions. See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for a list of simplified mnemonic examples.

Table 6-58. Branch Instructions

Name	Mnemonic	Syntax
Branch	**b (ba bl bla)**	target_addr
Branch Conditional	**bc (bca bcl bcla)**	BO,BI,target_addr
Branch Conditional to Link Register	**bclr (bclrl)**	BO,BI
Branch Conditional to Count Register	**bcctr (bcctrl)**	BO,BI

6.3.4.4.3 Condition Register Logical Instructions

Condition register logical instructions, shown in Table 6-59, and the Move Condition Register Field (**mcrf**) instruction are also defined as flow control instructions.

Table 6-59. Condition Register Logical Instructions

Name	Mnemonic	Syntax
Condition Register AND	**crand**	**crb**D,**crb**A,**crb**B
Condition Register OR	**cror**	**crb**D,**crb**A,**crb**B
Condition Register XOR	**crxor**	**crb**D,**crb**A,**crb**B
Condition Register NAND	**crnand**	**crb**D,**crb**A,**crb**B
Condition Register NOR	**crnor**	**crb**D,**crb**A,**crb**B
Condition Register Equivalent	**creqv**	**crb**D,**crb**A, **crb**B
Condition Register AND with Complement	**crandc**	**crb**D,**crb**A, **crb**B
Condition Register OR with Complement	**crorc**	**crb**D,**crb**A, **crb**B
Move Condition Register Field	**mcrf**	**crf**D,**crf**S

Note that if the LR update option is enabled for any of these instructions, the PowerPC architecture defines these forms of the instructions as invalid.

6.3.4.4.4 Trap Instructions

The trap instructions shown in Table 6-60 are provided to test for a specified set of conditions. If any of the conditions tested by a trap instruction are met, the system trap type program interrupt is taken. For more information, see the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual*. If the tested conditions are not met, instruction execution continues normally.

Table 6-60. Trap Instructions

Name	Mnemonic	Syntax
Trap Word Immediate	**twi**	TO,**r**A,SIMM
Trap Word	**tw**	TO,**r**A,**r**B

See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for a complete set of simplified mnemonics.

6.3.4.5 System Linkage Instruction—UISA

The System Call (**sc**) instruction permits a program to call on the system to perform a service; see Table 6-61 and also Section 6.3.6.1, “System Linkage Instructions—OEA,” for additional information.

Table 6-61. System Linkage Instruction—UISA

Name	Mnemonic	Syntax
System Call	**sc**	—

Executing this instruction causes the system call interrupt handler to be evoked. For more information, see the “Interrupts” chapter of the *e600 PowerPC Core Reference Manual*.

6.3.4.6 Processor Control Instructions—UISA

Processor control instructions are used to read from and write to the condition register (CR), machine state register (MSR), and special-purpose registers (SPRs). See Section 6.3.5.1, “Processor Control Instructions—VEA,” for the **mftb** instruction and Section 6.3.6.2, “Processor Control Instructions—OEA,” for information about the instructions used for reading from and writing to the MSR and SPRs.

6.3.4.6.1 Move To/From Condition Register Instructions

Table 6-62 summarizes the instructions for reading from or writing to the condition register.

Table 6-62. Move To/From Condition Register Instructions

Name	Mnemonic	Syntax
Move to Condition Register Fields	**mtcrf**	CRM,**r**S
Move to Condition Register from XER	**mcrxr**	**crf**D
Move from Condition Register	**mfcr**	**r**D

Implementation Note—The PowerPC architecture indicates that in some implementations the Move to Condition Register Fields (**mtcrf**) instruction can perform more slowly when only a portion of the fields are updated as opposed to all of the fields. The condition register access latency for the e600 core is the same in both cases, if multiple fields are affected. Note that an **mtcrf** to a single field is handled in the IU1s and latency may be lower if an **mtcrf** multi is split into its component single field pieces by the compiler.

6.3.4.6.2 Move To/From Special-Purpose Register Instructions—UISA

Table 6-63 lists the **mtspr** and **mfspr** instructions.

Table 6-63. Move To/From Special-Purpose Register Instructions (UISA)

Name	Mnemonic	Syntax
Move to Special-Purpose Register	**mtspr**	SPR,**r**S
Move from Special-Purpose Register	**mfspr**	**r**D,SPR

Table 6-64 lists the SPR numbers for user-level PowerPC SPR accesses.

Encodings for the e600-specific user-level SPRs are listed in Table 6-65.

Table 6-64. User-Level PowerPC SPR Encodings

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
CTR	9	00000	01001	User (UISA)	Both
LR	8	00000	01000	User (UISA)	Both
TBL[2]	268	01000	01100	User (VEA)	**mftb**
TBU [2]	269	01000	01101	User (VEA)	**mftb**
VRSAVE[3]	256	01000	00000	User (AltiVec/UISA)	Both
XER	1	00000	00001	User (UISA)	Both

1 Note that the order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.

2 The TB registers are referred to as TBRs rather than SPRs and can be written to using the **mtspr** instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using either the **mftb** instruction and specifying TBR 268 for TBL and TBR 269 for TBU.

3 Register defined by the AltiVec technology.

Table 6-65. User-Level SPR Encodings for e600-Defined Registers

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
UMMCR0	936	11101	01000	User	**mfspr**
UMMCR1	940	11101	01100	User	**mfspr**
UMMCR2	928	11101	00000	User	**mfspr**
UPMC1	937	11101	01001	User	**mfspr**
UPMC2	938	11101	01010	User	**mfspr**
UPMC3	941	11101	01101	User	**mfspr**

Table 6-65. User-Level SPR Encodings for e600-Defined Registers (continued)

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
UPMC4	942	11101	01110	User	**mfspr**
UPMC5	929	11101	00001	User	**mfspr**
UPMC6	930	11101	00010	User	**mfspr**
USIAR	939	11101	01011	User	**mfspr**

1 Note that the order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding. For mtspr and mfspr instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.

6.3.4.7 Memory Synchronization Instructions—UISA

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory access mechanisms. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for additional information about these instructions and about related aspects of memory synchronization. See Table 6-66 for a summary.

Table 6-66. Memory Synchronization Instructions—UISA

Name	Mnemonic	Syntax	Implementation Notes
Load Word and Reserve Indexed	**lwarx** [1]	**r**D,**r**A,**r**B	Programmers can use **lwarx** with **stwcx.** to emulate common semaphore operations such as test and set, compare and swap, exchange memory, and fetch and add. Both instructions must use the same EA. Reservation granularity is implementation-dependent. The e600 core makes reservations on behalf of aligned 32-byte sections of the memory address space. Executing **lwarx** and **stwcx.** to a page marked write-through (WIMG = 10xx) or caching-inhibited (WIMG = x1xx) or when the data cache is disabled or locked causes a DSI interrupt. If the location is not word-aligned, an alignment interrupt occurs. The **stwcx.** instruction is the only load/store instruction with a valid form if Rc is set. If Rc is zero, executing **stwcx.** sets CR0 to an undefined value.
Store Word Conditional Indexed	**stwcx.**[1]	**r**S,**r**A,**r**B	
Synchronize	**sync**	—	Because it delays execution of subsequent instructions until all previous instructions complete to where they cannot cause an interrupt, **sync** is a barrier against store gathering. Additionally, all load/store cache/MPX bus activities initiated by prior instructions are completed. Touch load operations (**dcbt**, **dcbtst**) must complete address translation, but need not complete on the MPX bus. The **sync** completes after a successful broadcast on the MPX bus. The latency of **sync** depends on the processor state when it is dispatched and on various system-level situations. Note that frequent use of **sync** will degrade performance.

1 Note that the e600 core implements **lwarx** and **stwcx.** as defined for embedded processors by the PowerPC ISA. The execution of an **lwarx** or **stwcx.** instruction to memory marked write-through or cache-inhibited causes a DSI interrupt.

Integrated devices with additional caches should take special care to recognize the hardware signaling caused by a SYNC MPX bus operation and perform the appropriate actions to guarantee that memory references that can be queued internally to the additional cache have been performed globally.

See Section 6.3.5.2, “Memory Synchronization Instructions—VEA,” for details about additional memory synchronization (**eieio**) instructions.

In the PowerPC architecture, the Rc bit must be zero for most load and store instructions. If Rc is set, the instruction form is invalid for **sync** and **lwarx** instructions. If the e600 core encounters one of these invalid instruction forms, it sets CR0 to an undefined value.

6.3.5 PowerPC VEA Instructions

The PowerPC virtual environment architecture (VEA) describes the semantics of the memory model that can be assumed by software processes, and includes descriptions of the cache model, cache control instructions, address aliasing, and other related issues. Implementations that conform to the VEA also adhere to the UISA, but do not necessarily adhere to the OEA.

This section describes additional instructions that are provided by the VEA.

6.3.5.1 Processor Control Instructions—VEA

In addition to the move to condition register instructions (specified by the UISA), the VEA defines the **mftb** instruction (user-level instruction) for reading the contents of the time base register; see the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*. for more information. Table 6-67 shows the **mftb** instruction.

Table 6-67. Move From Time Base Instruction

Name	Mnemonic	Syntax
Move from Time Base	**mftb**	**r**D, TBR

Simplified mnemonics are provided for the **mftb** instruction so it can be coded with the TBR name as part of the mnemonic rather than requiring it to be coded as an operand. See Appendix F, “Simplified Mnemonics,” in the *Programming Environments Manual* for simplified mnemonic examples and for simplified mnemonics for Move from Time Base (**mftb**) and Move from Time Base Upper (**mftbu**), which are variants of the **mftb** instruction rather than of **mfspr**. The **mftb** instruction serves as both a basic and simplified mnemonic. Assemblers recognize an **mftb** mnemonic with two operands as the basic form, and an **mftb** mnemonic with one operand as the simplified form.

Implementation Note—In the e600 core, note the following:

- The e600 core allows user-mode read access to the time base counter through the use of the Move from Time Base (**mftb**) instruction. As a 32-bit implementation of the PowerPC architecture, the e600 core can access TBU and TBL separately only.
- The time base counter is clocked at a frequency that is one-fourth that of the bus clock. Counting is enabled by assertion of the time base enable (*tben*) input signal.

6.3.5.2 Memory Synchronization Instructions—VEA

Memory synchronization instructions control the order in which memory operations are completed with respect to asynchronous events, and the order in which memory operations are seen by other processors or memory access mechanisms. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for more information about these instructions and about related aspects of memory synchronization.

In addition to the **sync** instruction (specified by UISA), the VEA defines the Enforce In-Order Execution of I/O (**eieio**) and Instruction Synchronize (**isync**) instructions. The number of cycles required to complete an **eieio** instruction depends on system parameters and on the core’s state when the instruction is issued. As a result, frequent use of this instruction can degrade performance. Note that the broadcast of these instructions on the MPX bus is controlled by the HID1[SYNCBE] bit.

Table 6-68 describes the memory synchronization instructions defined by the VEA.

Table 6-68. Memory Synchronization Instructions—VEA

Name	Mnemonic	Syntax	Implementation Notes
Enforce In-Order Execution of I/O	**eieio**	—	The **eieio** instruction is dispatched to the LSU and executes after all previous cache-inhibited or write-through accesses are performed; all subsequent instructions that generate such accesses execute after **eieio**. As the **eieio** operation does not affect the caches, it bypasses the L2 cache and is forwarded to the MPX bus. An EIEIO operation is broadcast on the MPX bus to enforce ordering in the memory system external to the core. Because the e600 core does reorder noncacheable accesses, **eieio** may be needed to force ordering. However, if store gathering is enabled and an **eieio** is detected in a store queue, stores are not gathered. Broadcasting **eieio** prevents other peripherals, such as bus bridge chips, from gathering stores.
Instruction Synchronize	**isync**	—	The **isync** instruction is refetch serializing; that is, it causes the e600 core to wait for all prior instructions to complete first then executes, purging all instructions from the core and then refetching the next instruction. The **isync** instruction is not executed until all previous instructions complete to the point where they cannot cause an interrupt. The **isync** instruction does not wait for all pending stores in the store queue to complete. Any instruction after an **isync** sees all effects of prior instructions occurring before the **isync**.

6.3.5.3 Memory Control Instructions—VEA

Memory control instructions can be classified as follows:

- Cache management instructions (user-level and supervisor-level)
- Translation lookaside buffer management instructions (OEA)

This section describes the user-level cache management instructions defined by the VEA. See Section 6.3.6.3, “Memory Control Instructions—OEA,” for information about supervisor-level cache, segment register manipulation, and translation lookaside buffer management instructions. For a complete description of the MPX bus operations caused by cache control instructions, see the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*.

6.3.5.3.1 User-Level Cache Instructions—VEA

The instructions summarized in this section help user-level programs manage caches within the core if they are implemented. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference*

Manual. for more information about cache topics. The following sections describe how these operations are treated with respect to the e600 core's caches.

As with other memory-related instructions, the effects of cache management instructions on memory are weakly-ordered. If the programmer must ensure that cache or other instructions have been performed with respect to all other processors and system mechanisms, a **sync** instruction must be placed after those instructions.

Note that the e600 core interprets cache control instructions (**icbi**, **dcbi**, **dcbf**, **dcbz**, and **dcbst**) as if they pertain only to the local L1 and L2 caches. A **dcbz** (with M set) is always broadcast on the core interface if it does not hit as modified in any core cache.

All cache control instructions to direct-store space function as no-ops. For information on how cache control instructions affect the L2 cache, see the "L1 and L2 Cache Operation" chapter of the *e600 PowerPC Core Reference Manual*.

The L1 cache can be flushed (if data is not shared and you need only to invalidate the cache) with the HID0[DCFI] and HID0[ICFI] bits. The L2 flush bit is L2CR[l2HWF] and the L2 invalidate bit is L2CR[L2I]. See the "L1 and L2 Cache Operation" chapter of the e600 PowerPC Core Reference Manual for more information.

Table 6-69 summarizes the cache instructions defined by the VEA. Note that these instructions are accessible to user-level programs.

Table 6-69. User-Level Cache Instructions

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Touch [1]	**dcbt**	**r**A,**r**B	The VEA defines this instruction to allow for potential system performance enhancements through the use of software-initiated prefetch hints. Implementations are not required to take any action based on execution of this instruction, but they can prefetch the cache block corresponding to the EA into their cache. When **dcbt** executes, the e600 core checks for protection violations (as for a load instruction). This instruction is treated as a no-op for the following cases: • The access causes a protection violation. • The page is mapped cache-inhibited or direct-store (T = 1). • The cache is locked or disabled • HID0[NOPTI] = 1 Otherwise, if no data is in the cache location, the e600 core requests a cache line fill. Data brought into the cache is validated as if it were a load instruction. The memory reference of a **dcbt** sets the referenced bit.

Table 6-69. User-Level Cache Instructions (continued)

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Touch for Store [1]	**dcbtst**	**r**A,**r**B	This instruction **dcbtst** can be no-oped by setting HID0[NOPTI]. The **dcbtst** instruction behaves similarly to a **dcbt** instruction, except that the line fill request on the MPX bus is signaled as read or read-claim, and the data is marked as exclusive in the L1 data cache if there is no shared response on the MPX bus. More specifically, the following cases occur depending on where the line currently exists or does not exist in the e600 core. • **dcbtst** hits in the L1 data cache. In this case, the **dcbtst** does nothing and the state of the line in the cache is not changed. Thus, if the line was in the shared state, a subsequent store hits on this shared line and incurs the associated latency penalties. • **dcbtst** misses in the L1 data cache and hits in the L2 cache. In this case, the **dcbtst** will reload the L1 data cache with the state found in the L2 cache. Again, if the line was in the shared state in the L2, a subsequent store will hit on this shared line and incur the associated latency penalties. • **dcbtst** misses in L1 data cache and L2 cache. In this case, the e600 core will request the line from memory with read or read-claim and reload the L1 data cache in the exclusive state. A subsequent store will hit on exclusive and can perform the store to the L1 data cache immediately. In addition, a **dcbtst** instruction will be no-oped if the target address of the **dcbtst** is mapped as write-through.
Data Cache Block Set to Zero	**dcbz**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations. For cache hits, 32 bytes of zeros are written to the cache block and the tag is marked modified. For cache misses with the replacement block marked not modified, the zero reload is performed and the cache block is marked modified. However, if the replacement block is marked modified, the contents are written back to memory first. The instruction takes an alignment interrupt if the cache block is locked or disabled or if the cache is marked WT or CI. If WIMG = xx1x (coherency enforced), the address is broadcast to the MPX bus before the zero reload fill. The interrupt priorities (from highest to lowest) are as follows: 1 Cache disabled—Alignment interrupt 2 Cache is locked—Alignment interrupt 3 Page marked write-through or cache-inhibited—alignment interrupt 4 BAT protection violation—DSI interrupt 5 TLB protection violation—DSI interrupt **dcbz** is broadcast if WIMG = xx1x (coherency enforced).
Data Cache Block Allocate	**dcba**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations. For cache hits, 32 bytes of zeros are written to the cache block and the tag is marked modified. For cache misses with the replacement block marked non-dirty, the zero reload is performed and the cache block is marked modified. However, if the replacement block is marked modified, the contents are written back to memory first. The instruction performs a no-op if the cache is locked or disabled or if the cache is marked WT or CI. If WIMG =xx1x (coherency enforced), the address is broadcast to the MPX bus before the zero reload fill. A no-op occurs for the following: • Cache is disabled • Cache is locked • Page marked write-through or cache-inhibited • BAT protection violation • TLB protection violation **dcba** is broadcast if WIMG = xx1x (coherency enforced).

Table 6-69. User-Level Cache Instructions (continued)

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Store	**dcbst**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations. • For cache hits with the tag marked not modified, no further action is taken. • For cache hits with the tag marked modified, the cache block is written back to memory and marked exclusive. If WIMG = xx1x (coherency enforced), **dcbst** is broadcast. The instruction acts like a load with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked. The interrupt priorities (from highest to lowest) for **dcbst** are as follows: 1 BAT protection violation—DSI interrupt 2 TLB protection violation—DSI interrupt
Data Cache Block Flush	**dcbf**	**r**A,**r**B	The EA is computed, translated, and checked for protection violations: • For cache hits with the tag marked modified, the cache block is written back to memory and the cache entry is invalidated. • For cache hits with the tag marked not modified, the entry is invalidated. • For cache misses, no further action is taken. A **dcbf** is broadcast if WIMG = xx1x (coherency enforced).The instruction acts like a load with respect to address translation and memory protection. It executes regardless of whether the cache is disabled or locked. The interrupt priorities (from highest to lowest) for **dcbf** are as follows: 1 BAT protection violation—DSI interrupt 2 TLB protection violation—DSI interrupt
Instruction Cache Block Invalidate	**icbi**	**r**A,**r**B	This instruction is broadcast on the MPX bus if WIMG = xx1x. **icbi** should always be followed by a **sync** and an **isync** to make sure that the effects of the **icbi** are seen by the instruction fetches following the **icbi** itself.

[1] A program that uses **dcbt** and **dcbtst** instructions improperly performs less efficiently. To improve performance, HID0[NOPTI] can be set, which causes dcbt and dcbtst to be no-oped at the cache. They do not cause MPX bus activity and cause only a 1-clock execution latency. The default state of this bit is zero which enables the use of these instructions.

6.3.5.4 Optional External Control Instructions

The PowerPC architecture defines an optional external control feature that, if implemented, is supported by the two external control instructions, **eciwx** and **ecowx**. These instructions are not implemented on the e600 core. Note that the EAR, which is accessible only in supervisor mode, must not be enabled. Otherwise, attempted execution of **eciwx**/**ecowx** causes boundedly undefined results.

6.3.6 PowerPC OEA Instructions

The PowerPC operating environment architecture (OEA) includes the structure of the memory management model, supervisor-level registers, and the interrupt model. Implementations that conform to the OEA also adhere to the UISA and the VEA. This section describes the instructions provided by the OEA.

6.3.6.1 System Linkage Instructions—OEA

This section describes the system linkage instructions (see Table 6-70). The user-level **sc** instruction lets a user program call on the system to perform a service and causes the processor to take a system call interrupt. The supervisor-level **rfi** instruction is used for returning from an interrupt handler.

Table 6-70. System Linkage Instructions—OEA

Name	Mnemonic	Syntax	Implementation Notes
System Call	**sc**	—	The **sc** instruction is context-synchronizing.
Return from Interrupt	**rfi**	—	**rfi** is context-synchronizing. For the e600 core, this means that **rfi** works its way to the final stage of the execution pipeline, updates architected registers, and redirects the instruction flow.

6.3.6.2 Processor Control Instructions—OEA

The instructions listed in Table 6-71 provide access to the segment registers for 32-bit implementations. These instructions operate completely independently of the MSR[IR] and MSR[DR] bit settings. Refer to “Synchronization Requirements for Special Registers and for Lookaside Buffers,” in Chapter 2, “Register Set,” of the *Programming Environments Manual* for serialization requirements and other recommended precautions to observe when manipulating the segment registers.

Table 6-71. Segment Register Manipulation Instructions (OEA)

Name	Mnemonic	Syntax	Implementation Notes
Move to Segment Register	**mtsr**	SR,**r**S	—
Move to Segment Register Indirect	**mtsrin**	**r**S,**r**B	—
Move from Segment Register	**mfsr**	**r**D,SR	—
Move from Segment Register Indirect	**mfsrin**	**r**D,**r**B	—

The processor control instructions used to access the MSR and the SPRs are discussed in this section. Table 6-72 lists instructions for accessing the MSR.

Table 6-72. Move To/From Machine State Register Instructions

Name	Mnemonic	Syntax
Move to Machine State Register	**mtmsr**	**r**S
Move from Machine State Register	**mfmsr**	**r**D

The OEA defines encodings of **mtspr** and **mfspr** to provide access to supervisor-level registers. The instructions are listed in Table 6-73.

Table 6-73. Move To/From Special-Purpose Register Instructions (OEA)

Name	Mnemonic	Syntax
Move to Special-Purpose Register	**mtspr**	SPR,**r**S
Move from Special-Purpose Register	**mfspr**	**r**D,SPR

Encodings for the architecture-defined SPRs are listed in Table 6-64. Encodings for e600-specific, supervisor-level SPRs are listed in Table 6-65. Simplified mnemonics are provided for **mtspr** and **mfspr** in Appendix F, "Simplified Mnemonics," in the *Programming Environments Manual*.

Table 6-74 lists the SPR numbers for supervisor-level PowerPC SPR accesses.

Table 6-74. Supervisor-Level PowerPC SPR Encodings

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
DABR[2]	1013	11111	10101	Supervisor (OEA)	Both
DAR	19	00000	10011	Supervisor (OEA)	Both
DBAT0L	537	10000	11001	Supervisor (OEA)	Both
DBAT0U	536	10000	11000	Supervisor (OEA)	Both
DBAT1L	539	10000	11011	Supervisor (OEA)	Both
DBAT1U	538	10000	11010	Supervisor (OEA)	Both
DBAT2L	541	10000	11101	Supervisor (OEA)	Both
DBAT2U	540	10000	11100	Supervisor (OEA)	Both
DBAT3L	543	10000	11111	Supervisor (OEA)	Both
DBAT3U	542	10000	11110	Supervisor (OEA)	Both
DEC	22	00000	10110	Supervisor (OEA)	Both
DSISR	18	00000	10010	Supervisor (OEA)	Both
EAR [2]	282	01000	11010	Supervisor (OEA)	Both
IBAT0L	529	10000	10001	Supervisor (OEA)	Both
IBAT0U	528	10000	10000	Supervisor (OEA)	Both
IBAT1L	531	10000	10011	Supervisor (OEA)	Both
IBAT1U	530	10000	10010	Supervisor (OEA)	Both
IBAT2L	533	10000	10101	Supervisor (OEA)	Both
IBAT2U	532	10000	10100	Supervisor (OEA)	Both
IBAT3L	535	10000	10111	Supervisor (OEA)	Both
IBAT3U	534	10000	10110	Supervisor (OEA)	Both
MMCR0 [2]	952	11101	11000	Supervisor (OEA)	Both
MMCR1 [2]	956	11101	11100	Supervisor (OEA)	Both
PIR [2]	1023	11111	11111	Supervisor (OEA)	Both
PMC1 [2]	953	11101	11001	Supervisor (OEA)	Both
PMC2 [2]	954	11101	11010	Supervisor (OEA)	Both
PMC3 [2]	957	11101	11101	Supervisor (OEA)	Both
PMC4 [2]	958	11101	11110	Supervisor (OEA)	Both

Table 6-74. Supervisor-Level PowerPC SPR Encodings (continued)

Register Name	SPR[1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
PMC5 [2]	945	11101	10001	Supervisor (OEA)	Both
PMC6 [2]	946	11101	10010	Supervisor (OEA)	Both
PVR	287	01000	11111	Supervisor (OEA)	**mfspr**
SDR1	25	00000	11001	Supervisor (OEA)	Both
SIAR [2]	955	11101	11011	Supervisor (OEA)	Both
SPRG0	272	01000	10000	Supervisor (OEA)	Both
SPRG1	273	01000	10001	Supervisor (OEA)	Both
SPRG2	274	01000	10010	Supervisor (OEA)	Both
SPRG3	275	01000	10011	Supervisor (OEA)	Both
SRR0	26	00000	11010	Supervisor (OEA)	Both
SRR1	27	00000	11011	Supervisor (OEA)	Both
TBL[3]	284	01000	11100	Supervisor (OEA)	**mtspr**
TBU [3]	285	01000	11101	Supervisor (OEA)	**mtspr**

1 Note that the order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.

2 Optional register defined by the PowerPC architecture.

3 The TB registers are referred to as TBRs rather than SPRs and can be written to using the **mtspr** instruction in supervisor mode and the TBR numbers here. The TB registers can be read in user mode using the **mftb** instruction and specifying TBR 268 for TBL and TBR 269 for TBU.

Encodings for the supervisor-level e600-specific SPRs are listed in Table 6-65.

Table 6-75. Supervisor-Level SPR Encodings for e600-Defined Registers

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
BAMR	951	11101	10111	Supervisor (OEA)	Both
DBAT4L[2]	569	10001	11001	Supervisor (OEA)	Both
DBAT4U [2]	568	10001	11000	Supervisor (OEA)	Both
DBAT5L [2]	571	10001	11011	Supervisor (OEA)	Both
DBAT5U [2]	570	10001	11010	Supervisor (OEA)	Both
DBAT6L [2]	573	10001	11101	Supervisor (OEA)	Both
DBAT6U [2]	572	10001	11100	Supervisor (OEA)	Both

Table 6-75. Supervisor-Level SPR Encodings for e600-Defined Registers (continued)

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
DBAT7L [2]	575	10001	11111	Supervisor (OEA)	Both
DBAT7U [2]	574	10001	11110	Supervisor (OEA)	Both
HID0	1008	11111	10000	Supervisor (OEA)	Both
HID1	1009	11111	10001	Supervisor (OEA)	Both
IABR	1010	11111	10010	Supervisor (OEA)	Both
IBAT4L [2]	561	10001	10001	Supervisor (OEA)	Both
IBAT4U [2]	560	10001	10000	Supervisor (OEA)	Both
IBAT5L [2]	563	10001	10011	Supervisor (OEA)	Both
IBAT5U [2]	562	10001	10010	Supervisor (OEA)	Both
IBAT6L [2]	565	10001	10101	Supervisor (OEA)	Both
IBAT6U [2]	564	10001	10100	Supervisor (OEA)	Both
IBAT7L [2]	567	10001	10111	Supervisor (OEA)	Both
IBAT7U [2]	566	10001	10110	Supervisor (OEA)	Both
ICTC	1019	11111	11011	Supervisor (OEA)	Both
ICTRL	1011	11111	10011	Supervisor (OEA)	Both
L2CR	1017	11111	11001	Supervisor (OEA)	Both
L2ERRADDR [2]	995	11111	00011	Supervisor (OEA)	**mfspr**
L2ERRATTR [2]	994	11111	00010	Supervisor (OEA)	Both
L2ERREADDR [2]	996	11111	00100	Supervisor (OEA)	**mfspr**
L2CAPTDATAHI [2]	988	11110	11100	Supervisor (OEA)	**mfspr**
L2CAPTDATALO [2]	989	11110	11101	Supervisor (OEA)	**mfspr**
L2CAPTECC [2]	990	11110	11110	Supervisor (OEA)	**mfspr**
L2ERRCTL [2]	997	11111	00101	Supervisor (OEA)	Both
L2ERRINJCTL [2]	987	11110	11011	Supervisor (OEA)	Both
L2ERRINJHI [2]	985	11110	11001	Supervisor (OEA)	Both
L2ERRINJLO [2]	986	11110	11010	Supervisor (OEA)	Both
L2ERRDET [2]	991	11110	11111	Supervisor (OEA)	Special [3]
L2ERRDIS [2]	992	11111	00000	Supervisor (OEA)	Both
L2ERRINTEN [2]	993	11111	00001	Supervisor (OEA)	Both
LDSTCR	1016	11111	1000	Supervisor (OEA)	Both
MMCR2	944	11101	10000	Supervisor (OEA)	Both

Table 6-75. Supervisor-Level SPR Encodings for e600-Defined Registers (continued)

Register Name	SPR [1]			Access	mfspr/mtspr
	Decimal	spr[5–9]	spr[0–4]		
MSSCR0	1014	11111	10110	Supervisor (OEA)	Both
MSSSR0	1015	11111	10111	Supervisor (OEA)	Both
PTEHI	981	11110	10101	Supervisor (OEA)	Both
PTELO	982	11110	10110	Supervisor (OEA)	Both
SPRG4 [2]	276	01000	10100	Supervisor (OEA)	Both
SPRG5 [2]	277	01000	10101	Supervisor (OEA)	Both
SPRG6 [2]	278	01000	100110	Supervisor (OEA)	Both
SPRG7 [2]	279	01000	10111	Supervisor (OEA)	Both
SVR	286	01000	11110	Supervisor (OEA)	**mfspr**
TLBMISS	980	11110	10100	Supervisor (OEA)	Both

1 Note that the order of the two 5-bit halves of the SPR number is reversed compared with actual instruction coding. For **mtspr** and **mfspr** instructions, the SPR number coded in assembly language does not appear directly as a 10-bit binary number in the instruction. The number coded is split into two 5-bit halves that are reversed in the instruction, with the high-order 5 bits appearing in bits 16–20 of the instruction and the low-order 5 bits in bits 11–15.

2 e600-specific register that may not be supported on other processors that implement the PowerPC architecture.

3 Most bits are bit reset/write 1 clear. A write of 0 to a bit does not change it. A write of 1 to a bit clears it. Reads act normally.

6.3.6.3 Memory Control Instructions—OEA

Memory control instructions include the following:

- Cache management instructions (supervisor-level and user-level)
- Translation lookaside buffer management instructions

This section describes supervisor-level memory control instructions. Section 6.3.5.3, “Memory Control Instructions—VEA,” describes user-level memory control instructions.

6.3.6.3.1 Supervisor-Level Cache Management Instruction—OEA

Table 6-76 lists the only supervisor-level cache management instruction.

Table 6-76. Supervisor-Level Cache Management Instruction

Name	Mnemonic	Syntax	Implementation Notes
Data Cache Block Invalidate	**dcbi**	**r**A,**r**B	The **dcbi** instruction is executed identically to the **dcbf** instruction except that it is privileged (supervisor-only). See Section 6.3.5.3.1, “User-Level Cache Instructions—VEA.”

See Section 6.3.5.3.1, “User-Level Cache Instructions—VEA,” for cache instructions that provide user-level programs the ability to manage the caches in the core. If the effective address references a direct-store segment, the instruction is treated as a no-op.

6.3.6.3.2 Translation Lookaside Buffer Management Instructions—OEA

The address translation mechanism is defined in terms of the segment descriptors and page table entries (PTEs) that processors use to locate the logical-to-physical address mapping for a particular access. These segment descriptors and PTEs reside in segment registers within the core and page tables in memory, respectively.

Implementation Note—The e600 core provides two implementation-specific instructions (**tlbld** and **tlbli**) that are used by software table search operations following TLB misses to load TLB entries within the core when HID0[STEN] = 1.

For more information on **tlbld** and **tlbli** refer to Section 6.3.8, "Implementation-Specific Instructions."

See the "Memory Management" chapter of the *e600 PowerPC Core Reference Manual* for more information about TLB operations. Table 6-77 summarizes the operation of the TLB instructions in the e600 core. Note that the broadcast of **tlbie** and **tlbsync** instructions is enabled by the setting of HID1[ABE].

Table 6-77. Translation Lookaside Buffer Management Instruction

Name	Mnemonic	Syntax	Implementation Notes
TLB Invalidate Entry	**tlbie**	**r**B	Invalidates both ways in both instruction and data TLB entries at the index provided by EA[14–19]. It executes regardless of the MSR[DR] and MSR[IR] settings. To invalidate all entries in both TLBs, the programmer should issue 64 **tlbie** instructions that each successively increment this field.
Load Data TLB Entry	**tlbld**	**r**B	Load Data TLB Entry Loads fields from the PTEHI and PTELO and the EA in **r**B to the way defined in **r**B[31].
Load Instruction TLB Entry	**tlbli**	**r**B	Load Instruction TLB Entry Loads fields from the PTEHI and PTELO and the EA in **r**B to the way defined in **r**B[31].
TLB Synchronize	**tlbsync**	—	TLBSYNC is broadcast.

Implementation Note—The **tlbia** instruction is optional for an implementation if its effects can be achieved through some other mechanism. Therefore, it is not implemented on the e600 core. As described above, **tlbie** can be used to invalidate a particular index of the TLB based on EA[14–19]—a sequence of 64 **tlbie** instructions followed by a **tlbsync** instruction invalidates all the TLB structures (for EA[14–19] = 0, 1, 2, . . . , 63). Attempting to execute **tlbia** causes an illegal instruction program interrupt.

The presence and exact semantics of the TLB management instructions are implementation-dependent. To minimize compatibility problems, system software should incorporate uses of these instructions into subroutines.

6.3.7 Recommended Simplified Mnemonics

The description of each instruction includes the mnemonic and a formatted list of operands. PowerPC-architecture-compliant assemblers support the mnemonics and operand lists. To simplify assembly language programming, a set of simplified mnemonics and symbols is provided for some of the most frequently-used instructions; refer to Appendix F, "Simplified Mnemonics," in the *Programming*

Environments Manual for a complete list. Programs written to be portable across the various assemblers for the PowerPC architecture should not assume the existence of mnemonics not described in this document.

6.3.8 Implementation-Specific Instructions

This section provides the details for the two e600 core implementation-specific instructions—**tlbld** and **tlbli**.

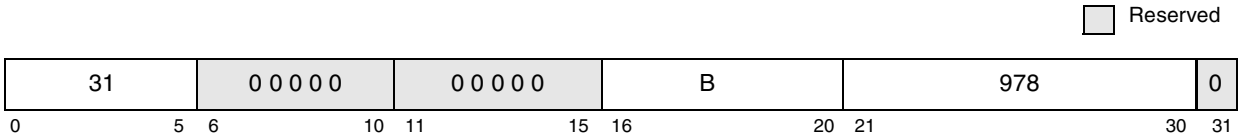
tlbld

tlbld

Load Data TLB Entry

Integer Unit

tlbld **r**B




```
EA ← (rB)
TLB entry created from PTEHI and PTELO
DTLB entry selected by EA[14–19] and rB[31] ← created TLB entry
```

The EA is the contents of **r**B. The **tlbld** instruction loads the contents of the PTEHI special purpose register and PTELO special purpose register into the selected data TLB entry. The set of the data TLB to be loaded is determined by EA[14–19]. The way to be loaded is determined by rB[31]. EA[10–13] are stored in the tag portion of the TLB and are used to match a new EA when a new EA is being translated.

The **tlbld** instruction should only be executed when address translation is disabled (MSR[IR] = 0 and MSR[DR] = 0).

Note that it is possible to execute the **tlbld** instruction when address translation is enabled; however, extreme caution should be used in doing so. If data address translation is enabled (MSR[DR] = 1), **tlbld** must be preceded by a **sync** instruction and succeeded by a context synchronizing instruction.

Note that if extended addressing is not enabled (HID0[XAEN] = 0), then PTELO[20–22] and PTELO[29] should be cleared (zero) by software when executing a **tlbld** instruction.

This is a supervisor-level instruction; it is also a e600-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

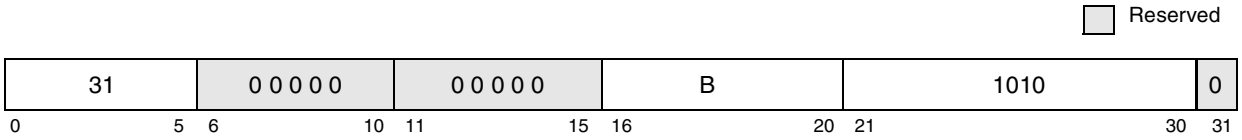
tlbli

tlbli

Load Instruction TLB Entry

Integer Unit

tlbli **r**B




```
EA ← (rB)
TLB entry created from PTEHI and PTELO
ITLB entry selected by EA[14–19] and rB[31] ← created TLB entry
```

The EA is the contents of **r**B. The **tlbli** instruction loads an instruction TLB entry. The **tlbli** instruction loads the contents of the PTEHI special purpose register and PTELO special purpose register into a selected instruction TLB entry. The set of the instruction TLB to be loaded is determined by EA[14–19]. The way to be loaded is determined by rB[31]. EA[10–13] are stored in the tag portion of the TLB and are used to match a new EA when a new EA is being translated.

The **tlbli** instruction should only be executed when address translation is disabled (MSR[IR] = 0 and MSR[DR] = 0).

Note that it is possible to execute the **tlbli** instruction when address translation is enabled; however, extreme caution should be used in doing so. If instruction address translation is enabled (MSR[IR] = 1), **tlbli** must be followed by a context synchronizing instruction such as **isync** or **rfi**.

Note that if extended addressing is not enabled (HID0[XAEN]=0) then PTELO[20–22] and PTELO[29] should be cleared (set to zero) by software when executing a **tlbli** instruction.

Note also that care should be taken to avoid modification of the instruction TLB entries that translate current instruction prefetch addresses.

This is a supervisor-level instruction; it is also a e600-specific instruction, and not part of the PowerPC instruction set.

Other registers altered:

- None

6.4 AltiVec Instructions

The following sections provide a general summary of the instructions and addressing modes defined by the AltiVec Instruction Set Architecture (ISA). For specific details on the AltiVec instructions see the *AltiVec Technology Programming Environments Manual* and the "AltiVec Technology Implementation" chapter of the *e600 PowerPC Core Reference Manual*." AltiVec instructions belong primarily to the UISA, unless otherwise noted. AltiVec instructions are divided into the following categories:

- Vector integer arithmetic instructions—These include arithmetic, logical, compare, rotate and shift instructions, described in Section 6.3.4.1, "Integer Instructions."
- Vector floating-point arithmetic instructions—These floating-point arithmetic instructions and floating-point modes are described in Section 6.3.4.2, "Floating-Point Instructions."
- Vector load and store instructions—These load and store instructions for vector registers are described in Section 6.5.3, "Vector Load and Store Instructions."
- Vector permutation and formatting instructions—These include pack, unpack, merge, splat, permute, select, and shift instructions, and are described in Section 6.5.5, "Vector Permutation and Formatting Instructions."
- Processor control instructions—These instructions are used to read and write from the AltiVec status and control register, and are described in Section 6.3.4.6, "Processor Control Instructions—UISA."
- Memory control instructions—These instructions are used for managing caches (user level and supervisor level), and are described in Section 6.6.1, "AltiVec Vector Memory Control Instructions—VEA."

This grouping of instructions does not necessarily indicate the execution unit that processes a particular instruction or group of instructions within a processor implementation.

Integer instructions operate on byte, half-word, and word operands. Floating-point instructions operate on single-precision operands. The AltiVec ISA uses instructions that are four bytes long and word-aligned. It provides for byte, half-word, word, and quad-word operand fetches and stores between memory and the vector registers (VRs).

Arithmetic and logical instructions do not read or modify memory. To use the contents of a memory location 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 to the target location using load and store instructions.

The AltiVec ISA supports both big-endian and little-endian byte ordering. The default byte and bit ordering is big-endian; see "Byte Ordering," in Chapter 3, "Operand Conventions," of the *AltiVec Technology Programming Environments Manual* for more information.

6.5 AltiVec UISA Instructions

This section describes the instructions defined in the AltiVec user instruction set architecture (UISA).

6.5.1 Vector Integer Instructions

The following are categories for vector integer instructions:

- Vector integer arithmetic instructions
- Vector integer compare instructions
- Vector integer logical instructions
- Vector integer rotate and shift instructions

Integer instructions use the content of VRs as source operands and also place results into VRs. Setting the Rc bit of a vector compare instruction causes the CR6 field of the PowerPC condition register (CR) to be updated; refer to Section 6.5.1.2, “Vector Integer Compare Instructions,” for more details.

The AltiVec integer instructions treat source operands as signed integers unless the instruction is explicitly identified as performing an unsigned operation. For example, both the Vector Add Unsigned Word Modulo (**vadduwm**) and Vector Multiply Odd Unsigned Byte (**vmuloub**) instructions interpret the operands as unsigned integers.

6.5.1.1 Vector Integer Arithmetic Instructions

Table 6-78 lists the integer arithmetic instructions for the processors that implement the PowerPC architecture.

Table 6-78. Vector Integer Arithmetic Instructions

Name	Mnemonic	Syntax
Vector Add Unsigned Integer [b,h,w] Modulo1	**vaddubm** **vadduhm** **vadduwm**	**v**D,**v**A,**v**B
Vector Add Unsigned Integer [b,h,w] Saturate	**vaddubs** **vadduhs** **vadduws**	**v**D,**v**A,**v**B
Vector Add Signed Integer [b.h.w] Saturate	**vaddsbs** **vaddshs** **vaddsws**	**v**D,**v**A,**v**B
Vector Add and Write Carry-out Unsigned Word	**vaddcuw**	**v**D,**v**A,**v**B
Vector Subtract Unsigned Integer Modulo	**vsububm** **vsubuhm** **vsubuwm**	**v**D,**v**A,**v**B
Vector Subtract Unsigned Integer Saturate	**vsububs** **vsubuhs** **vsubuws**	**v**D,**v**A,**v**B

Table 6-78. Vector Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Vector Subtract Signed Integer Saturate	**vsubsbs** **vsubshs** **vsubsws**	**v**D,**v**A,**v**B
Vector Subtract and Write Carry-out Unsigned Word	**vsubcuw**	**v**D,**v**A,**v**B
Vector Multiply Odd Unsigned Integer [b,h] Modulo	**vmuloub** **vmulouh**	**v**D,**v**A,**v**B
Vector Multiply Odd Signed Integer [b,h] Modulo	**vmulosb** **vmulosh**	**v**D,**v**A,**v**B
Vector Multiply Even Unsigned Integer [b,h] Modulo	**vmuleub** **vmuleuh**	**v**D,**v**A,**v**B
Vector Multiply Even Signed Integer [b,h] Modulo	**vmulesb** **vmulesh**	**v**D,**v**A,**v**B
Vector Multiply-High and Add Signed Half-Word Saturate	**vmhaddshs**	**v**D,**v**A,**v**B, **v**C
Vector Multiply-High Round and Add Signed Half-Word Saturate	**vmhraddshs**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Low and Add Unsigned Half-Word Modulo	**vmladduhm**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Unsigned Integer [b,h] Modulo	**vmsumubm** **vmsumuhm**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Signed Half-Word Saturate	**vmsumshs**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Unsigned Half-Word Saturate	**vmsumuhs**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Mixed Byte Modulo	**vmsummbm**	**v**D,**v**A,**v**B,**v**C
Vector Multiply-Sum Signed Half-Word Modulo	**vmsumshm**	**v**D,**v**A,**v**B,**v**C
Vector Sum Across Signed Word Saturate	**vsumsws**	**v**D,**v**A,**v**B
Vector Sum Across Partial (1/2) Signed Word Saturate	**vsum2sws**	**v**D,**v**A,**v**B
Vector Sum Across Partial (1/4) Unsigned Byte Saturate	**vsum4ubs**	**v**D,**v**A,**v**B
Vector Sum Across Partial (1/4) Signed Integer Saturate	**vsum4sbs** **vsum4shs**	**v**D,**v**A,**v**B
Vector Average Unsigned Integer	**vavgub** **vavguh** **vavguw**	**v**D,**v**A,**v**B
Vector Average Signed Integer	**vavgsb** **vavgsh** **vavgsw**	**v**D,**v**A,**v**B
Vector Maximum Unsigned Integer	**vmaxub** **vmaxuh** **vmaxuw**	**v**D,**v**A,**v**B
Vector Maximum Signed Integer	**vmaxsb** **vmaxsh** **vmaxsw**	**v**D,**v**A,**v**B

Table 6-78. Vector Integer Arithmetic Instructions (continued)

Name	Mnemonic	Syntax
Vector Minimum Unsigned Integer	**vminub** **vminuh** **vminuw**	**v**D,**v**A,**v**B
Vector Minimum Signed Integer	**vminsb** **vminsh** **vminsw**	**v**D,**v**A,**v**B

6.5.1.2 Vector Integer Compare Instructions

The vector integer compare instructions algebraically or logically compare the contents of the elements in vector register **v**A with the contents of the elements in **v**B. Each compare result vector is comprised of TRUE (0xFF, 0xFFFF, 0xFFFF_FFFF) or FALSE (0x00, 0x0000, 0x0000_0000) elements of the size specified by the compare source operand element (byte, half word, or word). The result vector can be directed to any VR and can be manipulated with any of the instructions as normal data (for example, combining condition results).

Vector compares provide equal-to and greater-than predicates. Others are synthesized from these by logically combining or inverting result vectors.

The integer compare instructions (shown in Table 6-80) can optionally set the CR6 field of the PowerPC condition register. If Rc = 1 in the vector integer compare instruction, then CR6 is set to reflect the result of the comparison, as follows in Table 6-79.

Table 6-79. CR6 Field Bit Settings for Vector Integer Compare Instructions

CR Bit	CR6 Bit	Vector Compare
24	0	1 Relation is true for all element pairs (that is, **v**D is set to all ones)
25	1	0
26	2	1 Relation is false for all element pairs (that is, register **v**D is cleared)
27	3	0

Table 6-80 summarizes the vector integer compare instructions.

Table 6-80. Vector Integer Compare Instructions

Name	Mnemonic	Syntax
Vector Compare Greater than Unsigned Integer	**vcmpgtub**[.] **vcmpgtuh**[.] **vcmpgtuw**[.]	**v**D,**v**A,**v**B
Vector Compare Greater than Signed Integer	**vcmpgtsb**[.] **vcmpgtsh**[.] **vcmpgtsw**[.]	**v**D,**v**A,**v**B
Vector Compare Equal to Unsigned Integer	**vcmpequb**[.] **vcmpequh**[.] **vcmpequw**[.]	**v**D,**v**A,**v**B

6.5.1.3 Vector Integer Logical Instructions

The vector integer logical instructions shown in Table 6-81 perform bit-parallel operations on the operands.

Table 6-81. Vector Integer Logical Instructions

Name	Mnemonic	Syntax
Vector Logical AND	**vand**	**v**D,**v**A,**v**B
Vector Logical OR	**vor**	**v**D,**v**A,**v**B
Vector Logical XOR	**vxor**	**v**D,**v**A,**v**B
Vector Logical AND with Complement	**vandc**	**v**D,**v**A,**v**B
Vector Logical NOR	**vnor**	**v**D,**v**A,**v**B

6.5.1.4 Vector Integer Rotate and Shift Instructions

The vector integer rotate instructions are summarized in Table 6-82.

Table 6-82. Vector Integer Rotate Instructions

Name	Mnemonic	Syntax
Vector Rotate Left Integer	**vrlb** **vrlh** **vrlw**	**v**D,**v**A,**v**B

The vector integer shift instructions are summarized in Table 6-83.

Table 6-83. Vector Integer Shift Instructions

Name	Mnemonic	Syntax
Vector Shift Left Integer	**vslb** **vslh** **vslw**	**v**D,**v**A,**v**B
Vector Shift Right Integer	**vsrb** **vsrh** **vsrw**	**v**D,**v**A,**v**B
Vector Shift Right Algebraic Integer	**vsrab** **vsrah** **vsraw**	**v**D,**v**A,**v**B

6.5.2 Vector Floating-Point Instructions

This section describes the vector floating-point instructions that include the following:

- Vector floating-point arithmetic instructions
- Vector floating-point rounding and conversion instructions
- Vector floating-point compare instructions
- Vector floating-point estimate instructions

The AltiVec floating-point data format complies with the ANSI/IEEE-754 standard as defined for single precision. A quantity in this format represents a signed normalized number, a signed denormalized number, a signed zero, a signed infinity, a quiet not a number (QNaN), or a signaling NaN (SNaN). Operations conform to the description in the section “AltiVec Floating-Point Instructions-UISA,” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual*.

The AltiVec ISA does not report IEEE exceptions but rather produces default results as specified by the Java/IEEE/C9X Standard; for further details on exceptions see “Floating-Point Exceptions,” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual*.

6.5.2.1 Vector Floating-Point Arithmetic Instructions

The floating-point arithmetic instructions are summarized in Table 6-84.

Table 6-84. Vector Floating-Point Arithmetic Instructions

Name	Mnemonic	Syntax
Vector Add Floating-Point	**vaddfp**	**v**D,**v**A,**v**B
Vector Subtract Floating-Point	**vsubfp**	**v**D,**v**A,**v**B
Vector Maximum Floating-Point	**vmaxfp**	**v**D,**v**A,**v**B
Vector Minimum Floating-Point	**vminfp**	**v**D,**v**A,**v**B

6.5.2.2 Vector Floating-Point Multiply-Add Instructions

Vector multiply-add instructions are critically important to performance because a multiply followed by a data dependent addition is the most common idiom in DSP algorithms. In most implementations, floating-point multiply-add instructions perform with the same latency as either a multiply or add alone, thus doubling performance in comparing to the otherwise serial multiply and adds.

AltiVec floating-point multiply-add instructions fuse (a multiply-add fuse implies that the full product participates in the add operation without rounding, only the final result rounds). This not only simplifies the implementation and reduces latency (by eliminating the intermediate rounding) but also increases the accuracy compared to separate multiply and adds.

The floating-point multiply-add instructions are summarized in Table 6-85.

Table 6-85. Vector Floating-Point Multiply-Add Instructions

Name	Mnemonic	Syntax
Vector Multiply-Add Floating-Point	**vmaddfp**	**v**D,**v**A,**v**C,**v**B
Vector Negative Multiply-Subtract Floating-Point	**vnmsubfp**	**v**D,**v**A,**v**C,**v**B

6.5.2.3 Vector Floating-Point Rounding and Conversion Instructions

All AltiVec floating-point arithmetic instructions use the IEEE Std. 754 default rounding mode round-to-nearest. The AltiVec ISA does not provide the IEEE Std. 754 directed rounding modes.

The AltiVec ISA provides separate instructions for converting floating-point numbers to integral floating-point values for all IEEE Std. 754 rounding modes as follows:

- Round-to-nearest (**vrfin**) (round)
- Round-toward-zero (**vrfiz**) (truncate)
- Round-toward-minus-infinity (**vrfim**) (floor)
- Round-toward-positive-infinity (**vrfip**) (ceiling)

Floating-point conversions to integers (**vctuxs**, **vctsxs**) use round-toward-zero (truncate) rounding. The floating-point rounding instructions are shown in Table 6-86.

Table 6-86. Vector Floating-Point Rounding and Conversion Instructions

Name	Mnemonic	Syntax
Vector Round to Floating-Point Integer Nearest	**vrfin**	**v**D,**v**B
Vector Round to Floating-Point Integer toward Zero	**vrfiz**	**v**D,**v**B
Vector Round to Floating-Point Integer toward Positive Infinity	**vrfip**	**v**D,**v**B
Vector Round to Floating-Point Integer toward Minus Infinity	**vrfim**	**v**D,**v**B
Vector Convert from Unsigned Fixed-Point Word	**vcfux**	**v**D,**v**B,UIMM
Vector Convert from Signed Fixed-Point Word	**vcfsx**	**v**D,**v**B,UIMM
Vector Convert to Unsigned Fixed-Point Word Saturate	**vctuxs**	**v**D,**v**B,UIMM
Vector Convert to Signed Fixed-Point Word Saturate	**vctsxs**	**v**D,**v**B,UIMM

6.5.2.4 Vector Floating-Point Compare Instructions

The floating-point compare instructions are summarized in Table 6-87.

Table 6-87. Vector Floating-Point Compare Instructions

Name	Mnemonic	Syntax
Vector Compare Greater Than Floating-Point [Record]	**vcmpgtfp**[.]	**v**D,**v**A,**v**B
Vector Compare Equal to Floating-Point [Record]	**vcmpeqfp**[.]	**v**D,**v**A,**v**B
Vector Compare Greater Than or Equal to Floating-Point [Record]	**vcmpgeqfp**[.]	**v**D,**v**A,**v**B
Vector Compare Bounds Floating-Point [Record]	**vcmpbfp**[.]	**v**D,**v**A,**v**B

6.5.2.5 Vector Floating-Point Estimate Instructions

The floating-point estimate instructions are summarized in Table 6-88.

Table 6-88. Vector Floating-Point Estimate Instructions

Name	Mnemonic	Syntax
Vector Reciprocal Estimate Floating-Point	**vrefp**	**v**D,**v**B
Vector Reciprocal Square Root Estimate Floating-Point	**vrsqrtefp**	**v**D,**v**B

Table 6-88. Vector Floating-Point Estimate Instructions (continued)

Name	Mnemonic	Syntax
Vector Log2 Estimate Floating-Point	**vlogefp**	**v**D,**v**B
Vector 2 Raised to the Exponent Estimate Floating-Point	**vexptefp**	**v**D,**v**B

6.5.3 Vector Load and Store Instructions

Only very basic load and store operations are provided in the AltiVec ISA. This keeps the circuitry in the memory path fast so the latency of memory operations is minimized. Instead, a powerful set of field manipulation instructions are provided to manipulate data into the desired alignment and arrangement after the data has been brought into the VRs.

Load vector indexed (**lvx**, **lvxl**) and store vector indexed (**stvx**, **stvxl**) instructions transfer an aligned quad-word vector between memory and VRs. Load vector element indexed (**lvebx**, **lvehx**, **lvewx**) and store vector element indexed instructions (**stvebx**, **stvehx**, **stvewx**) transfer byte, half-word, and word scalar elements between memory and VRs.

6.5.3.1 Vector Load Instructions

For vector load instructions, the byte, half-word, word, or quad-word addressed by the EA (effective address) is loaded into **v**D.

The default byte and bit ordering is big-endian as in the PowerPC architecture; see “Byte Ordering,” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual* for information about little-endian byte ordering.

Table 6-89 summarizes the vector load instructions.

Table 6-89. Vector Integer Load Instructions

Name	Mnemonic	Syntax
Load Vector Element Integer Indexed	**lvebx** **lvehx** **lvewx**	**v**D,**r**A,**r**B
Load Vector Element Indexed	**lvx**	**v**D,**r**A,**r**B
Load Vector Element Indexed LRU [1]	**lvxl**	**v**D,**r**A,**r**B

[1] On the e600 core, **lvxl** and **stvxl** are interpreted to be transient. See The “AltiVec Technology Implementation” chapter of the *e600 PowerPC Core Reference Manual*.”

6.5.3.2 Vector Load Instructions Supporting Alignment

The **lvsl** and **lvsr** instructions can be used to create the permute control vector to be used by a subsequent **vperm** instruction. Let X and Y be the contents of **v**A and **v**B specified by **vperm**. The control vector created by **lvsl** causes the **vperm** to select the high-order 16 bytes of the result of shifting the 32-byte value X || Y left by sh bytes (sh = the value in EA[60–63]). The control vector created by **lvsr** causes the **vperm** to select the low-order 16 bytes of the result of shifting X || Y right by sh bytes.

Table 6-90 summarizes the vector alignment instructions.

Table 6-90. Vector Load Instructions Supporting Alignment

Name	Mnemonic	Syntax
Load Vector for Shift Left	**lvsl**	**v**D,**r**A,**r**B
Load Vector for Shift Right	**lvsr**	**v**D,**r**A,**r**B

6.5.3.3 Vector Store Instructions

For vector store instructions, the contents of the VR used as a source (**v**S) are stored into the byte, half word, word or quad word in memory addressed by the effective address (EA). Table 6-91 provides a summary of the vector store instructions.

Table 6-91. Vector Integer Store Instructions

Name	Mnemonic	Syntax
Store Vector Element Integer Indexed	**svetbx** **svethx** **svetwx**	**v**S,**r**A,**r**B
Store Vector Element Indexed	**stvx**	**v**S,**r**A,**r**B
Store Vector Element Indexed LRU[1]	**stvxl**	**v**S,**r**A,**r**B

[1] On the e600 core, **lvxl** and **stvxl** are interpreted to be transient. See the "AltiVec Technology Implementation" chapter of the *e600 PowerPC Core Reference Manual*."

6.5.4 Control Flow

AltiVec instructions can be freely intermixed with existing PowerPC instructions to form a complete program. AltiVec instructions provide a vector compare and select mechanism to implement conditional execution as the preferred mechanism to control data flow in AltiVec programs. In addition, AltiVec vector compare instructions can update the condition register thus providing the communication from AltiVec execution units to PowerPC branch instructions necessary to modify program flow based on vector data.

6.5.5 Vector Permutation and Formatting Instructions

Vector pack, unpack, merge, splat, permute, select, and shift instructions can be used to accelerate various vector math operations and vector formatting. Details of these instructions follow.

6.5.5.1 Vector Pack Instructions

Half-word vector pack instructions (**vpkuhum**, **vpkuhus**, **vpkshus**, **vpkshss**) truncate the sixteen half words from two concatenated source operands producing a single result of sixteen bytes (quad word) using either modulo (2^8), 8-bit signed-saturation, or 8-bit unsigned-saturation to perform the truncation. Similarly, word vector pack instructions (**vpkuwum**, **vpkuwus**, **vpkswus**, **vpksws**) truncate the eight words from two concatenated source operands, producing a single result of eight half words using modulo (2^{16}), 16-bit signed-saturation, or 16-bit unsigned-saturation to perform the truncation.

Table 6-92 describes the vector pack instructions.

Table 6-92. Vector Pack Instructions

Name	Mnemonic	Syntax
Vector Pack Unsigned Integer [h,w] Unsigned Modulo	**vpkuhum** **vpkuwum**	**v**D, **v**A, **v**B
Vector Pack Unsigned Integer [h,w] Unsigned Saturate	**vpkuhus** **vpkuwus**	**v**D, **v**A, **v**B
Vector Pack Signed Integer [h,w] Unsigned Saturate	**vpkshus** **vpkswus**	**v**D, **v**A, **v**B
Vector Pack Signed Integer [h,w] signed Saturate	**vpkshss** **vpkswss**	**v**D, **v**A, **v**B
Vector Pack Pixel	**vpkpx**	**v**D, **v**A, **v**B

6.5.5.2 Vector Unpack Instructions

Byte vector unpack instructions unpack the 8 low bytes (or 8 high bytes) of one source operand into 8 half words using sign extension to fill the most-significant bytes (MSBs). Half word vector unpack instructions unpack the 4 low half words (or 4 high half words) of one source operand into 4 words using sign extension to fill the MSBs.

Two special purpose forms of vector unpack are provided—the Vector Unpack Low Pixel (**vupklpx**) and the Vector Unpack High Pixel (**vupkhpx**) instructions for 1/5/5/5 αRGB pixels. The 1/5/5/5 pixel vector unpack, unpacks the four low 1/5/5/5 pixels (or four 1/5/5/5 high pixels) into four 32-bit (8/8/8/8) pixels. The 1-bit α element in each pixel is sign extended to 8 bits, and the 5-bit R, G, and B elements are each zero extended to 8 bits.

Table 6-93 describes the unpack instructions.

Table 6-93. Vector Unpack Instructions

Name	Mnemonic	Syntax
Vector Unpack High Signed Integer	**vupkhsb** **vupkhsh**	**v**D, **v**B
Vector Unpack High Pixel	**vupkhpx**	**v**D, **v**B
Vector Unpack Low Signed Integer	**vupklsb** **vupklsh**	**v**D, **v**B
Vector Unpack Low Pixel	**vupklpx**	**v**D, **v**B

6.5.5.3 Vector Merge Instructions

Byte vector merge instructions interleave the 8 low bytes or 8 high bytes from two source operands producing a result of 16 bytes. Similarly, half-word vector merge instructions interleave the 4 low half words (or 4 high half words) of two source operands producing a result of 8 half words, and word vector merge instructions interleave the 2 low words or 2 high words from two source operands producing a result

of 4 words. The vector merge instruction has many uses. For example, it can be used to efficiently transpose SIMD vectors. Table 6-94 describes the merge instructions.

Table 6-94. Vector Merge Instructions

Name	Mnemonic	Syntax
Vector Merge High Integer	**vmrghb** **vmrghh** **vmrghw**	**v**D, **v**A, **v**B
Vector Merge Low Integer	**vmrglb** **vmrglh** **vmrglw**	**v**D, **v**A, **v**B

6.5.5.4 Vector Splat Instructions

When a program needs to perform arithmetic vector operations, the vector splat instructions can be used in preparation for performing arithmetic for which one source vector is to consist of elements that all have the same value. Vector splat instructions can be used to move data where it is required. For example to multiply all elements of a vector register (VR) by a constant, the vector splat instructions can be used to splat the scalar into the VR. Likewise, when storing a scalar into an arbitrary memory location, it must be splatted into a VR, and that VR must be specified as the source of the store. This guarantees that the data appears in all possible positions of that scalar size for the store.

Table 6-95. Vector Splat Instructions

Name	Mnemonic	Syntax
Vector Splat Integer	**vspltb** **vsplth** **vspltw**	**v**D, **v**B, UIMM
Vector Splat Immediate Signed Integer	**vspltisb** **vspltish** **vspltisw**	**v**D, SIMM

6.5.5.5 Vector Permute Instructions

Permute instructions allow any byte in any two source VRs to be directed to any byte in the destination vector. The fields in a third source operand specify from which field in the source operands the corresponding destination field is taken. The Vector Permute (**vperm**) instruction is a very powerful one that provides many useful functions. For example, it provides a way to perform table-lookups and data alignment operations. An example of how to use the **vperm** instruction in aligning data is described in “Quad-Word Data Alignment” in Chapter 3, “Operand Conventions,” of the *AltiVec Technology Programming Environments Manual*. Table 6-92 describes the vector permute instruction.

Table 6-96. Vector Permute Instruction

Name	Mnemonic	Syntax
Vector Permute	**vperm**	**v**D, **v**A,**v**B,**v**C

6.5.5.6 Vector Select Instruction

Data flow in the vector unit can be controlled without branching by using a vector compare and the Vector Select (**vsel**) instructions. In this use, the compare result vector is used directly as a mask operand to vector select instructions.The **vsel** instruction selects one field from one or the other of two source operands under control of its mask operand. Use of the TRUE/FALSE compare result vector with select in this manner produces a two instruction equivalent of conditional execution on a per-field basis. Table 6-97 describes the **vsel** instruction.

Table 6-97. Vector Select Instruction

Name	Mnemonic	Syntax
Vector Select	**vsel**	**v**D,**v**A,**v**B,**v**C

6.5.5.7 Vector Shift Instructions

The vector shift instructions shift the contents of one or of two VRs left or right by a specified number of bytes (**vslo**, **vsro**, **vsldoi**) or bits (**vsl**, **vsr**). Depending on the instruction, this shift count is specified either by low-order bits of a VR or by an immediate field in the instruction. In the former case the low-order 7 bits of the shift count register give the shift count in bits ($0 \le count \le 127$). Of these 7 bits, the high-order 4 bits give the number of complete bytes by which to shift and are used by **vslo** and **vsro**; the low-order 3 bits give the number of remaining bits by which to shift and are used by **vsl** and **vsr**.

Table 6-98 describes the vector shift instructions.

Table 6-98. Vector Shift Instructions

Name	Mnemonic	Syntax
Vector Shift Left	**vsl**	**v**D,**v**A,**v**B
Vector Shift Right	**vsr**	**v**D,**v**A,**v**B
Vector Shift Left Double by Octet Immediate	**vsldoi**	**v**D,**v**A,**v**B,SH
Vector Shift Left by Octet	**vslo**	**v**D,**v**A,**v**B
Vector Shift Right by Octet	**vsro**	**v**D,**v**A,**v**B

6.5.5.8 Vector Status and Control Register Instructions

Table 6-99 summarizes the instructions for reading from or writing to the AltiVec status and control register (VSCR), described in the “AltiVec Technology Implementation” chapter of the *e600 PowerPC Core Reference Manual*.”

Table 6-99. Move To/From VSCR Register Instructions

Name	Mnemonic	Syntax
Move to AltiVec Status and Control Register	**mtvscr**	**v**B
Move from AltiVec Status and Control Register	**mfvscr**	**v**B

6.6 AltiVec VEA Instructions

The PowerPC virtual environment architecture (VEA) describes the semantics of the memory model that can be assumed by software processes, and includes descriptions of the cache model, cache-control instructions, address aliasing, and other related issues. Implementations that conform to the VEA also adhere to the UISA, but may not necessarily adhere to the OEA. For further details, see Chapter 4, “Addressing Mode and Instruction Set Summary,” in the *Programming Environments Manual.*

This section describes the additional instructions that are provided by the AltiVec ISA for the VEA.

6.6.1 AltiVec Vector Memory Control Instructions—VEA

Memory control instructions include the following types:

- Cache management instructions (user-level and supervisor-level)
- Translation lookaside buffer (TLB) management instructions

This section briefly summarizes the user-level cache management instructions defined by the AltiVec VEA. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*. for more information about supervisor-level cache, segment register manipulation, and TLB management instructions.

The AltiVec architecture specifies the data stream touch instructions **dst(t)**, **dstst(t)**, and it specifies two data stream stop (**dss(all)**) instructions. The e600 core implements all of them. The term **dst***x* used below refers to all of the stream touch instructions.

The instructions summarized in this section provide user-level programs the ability to manage caches within the core. See the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual* for more information about cache topics.

Bandwidth between the processor and memory is managed explicitly by the programmer through the use of cache management instructions. These instructions provide a way for software to communicate to the cache hardware how it should prefetch and prioritize the writeback of data. The principal instruction for this purpose is a software-directed cache prefetch instruction called data stream touch (**dst**). Other related instructions are provided for complete control of the software-directed cache prefetch mechanism.

Table 6-100 summarizes the directed prefetch cache instructions defined by the AltiVec VEA. Note that these instructions are accessible to user-level programs.

Table 6-100. AltiVec User-Level Cache Instructions

Name	Mnemonic	Syntax	Implementation Notes
Data Stream Touch (non-transient)	**dst**	**r**A,**r**B,STRM	—
Data Stream Touch Transient	**dstt**	**r**A,**r**B,STRM	Used for last access
Data Stream Touch for Store	**dstst**	**r**A,**r**B,STRM	Not recommended for use in the e600 core
Data Stream Touch for Store Transient	**dststt**	**r**A,**r**B,STRM	Not recommended for use in the e600 core
Data Stream Stop (one stream)	**dss**	STRM	—
Data Stream Stop All	**dssall**	STRM	—

For detailed information for how to use these instructions, see the “AltiVec Technology Implementation” chapter of the *e600 PowerPC Core Reference Manual.*

6.6.2 AltiVec Instructions with Specific Implementations for the e600 Core

The AltiVec architecture specifies Load Vector Indexed LRU (**lvxl**) and Store Vector Indexed LRU (**stvxl**) instructions. The architecture suggests that these instructions differ from regular AltiVec load and store instructions in that they leave cache entries in a least recently used (LRU) state instead of a most recently used (MRU) state. This supports efficient processing of data which is known to have little reuse and poor caching characteristics. The e600 core implements these instructions as suggested. They follow all the cache allocation and replacement policies described in the “L1 and L2 Cache Operation” chapter of the *e600 PowerPC Core Reference Manual*, but they leave their addressed cache entries in the LRU state. In addition, all LRU instructions are also interpreted to be transient and are also treated as described in the “AltiVec Technology Implementation” chapter of the *e600 PowerPC Core Reference Manual.*

Part III
Memory, Peripherals, and I/O Interfaces

Part III defines the memory, peripherals, and I/O interfaces of the MPC8610 and it describes how these blocks interact with one another and with other blocks on the device. The following chapters are included:

- Chapter 7, "MPX Coherency Module (MCM) Overview," defines the e600 coherency module and how it facilitates communication between the e600 core complex and the other blocks that comprise the coherent memory domain of the MPC8610.
- Chapter 8, "DDR Memory Controller," describes the DDR SDRAM memory controller of the MPC8610.
- Chapter 9, "Enhanced Local Bus Controller," describes the enhanced local bus controller of the MPC8610. The enhanced local bus controller (eLBC) provides a seamless interface to many types of memory devices and peripherals.
- Chapter 10, "Display Interface Unit (DIU)," describes the integrated LCD controller of the MPC8610.
- Chapter 11, "Programmable Interrupt Controller (PIC)," describes the OpenPIC-compliant embedded programmable interrupt controller (PIC) of the MPC8610.
- Chapter 12, "I^2C Modules," describes the inter-IC (I^2C) bus controller of the MPC8610.
- Chapter 13, "DUART," describes the (dual) universal asynchronous receiver/transmitters (UARTs) which feature a PC16552D-compatible programming model.
- Chapter 14, "Fast/Serial Infrared Interfaces (FIRI/SIRI)," describes the IrDa-compatible infrared interface controller of the MPC8610.
- Chapter 15, "Serial Peripheral Interface," describes the serial peripheral interface (SPI) controller of the MPC8610.
- Chapter 16, "Synchronous Serial Interface (SSI)," describes the synchronous serial interface (SSI) controllers of the MPC8610 that implement the inter-IC sound bus standard (I^2S) and Intel AC97 standards for audio.
- Chapter 17, "Global Timer Module," describes the global timer modules of the MPC8610.
- Chapter 18, "Watchdog Timer," describes the watchdog timer of the MPC8610.
- Chapter 19, "DMA Controllers," describes the dual, four-channel general-purpose DMA controllers of the MPC8610.
- Chapter 20, "PCI Bus Interface," describes the PCI controller and provides a basic description of the PCI bus operations.
- Chapter 21, "PCI Express Interface Controller," describes the PCI Express controllers of this device.
- Chapter 22, "General Purpose I/O (GPIO)," describes the two general purpose I/O (GPIO) modules of the MPC8610.

Chapter 7
MPX Coherency Module (MCM) Overview

7.1 Introduction

The MPX coherency module provides a mechanism for I/O-initiated transactions to be snooped in order to maintain data coherency for cacheable address spaces. It has an MPX bus arbiter which supports core transactions and incoming transactions that must be snooped by the core's caches. These incoming transactions are from the PCI Express ports the DMA controllers, the DIU, and the boot sequencer.The MCM also provides a flexible switch structure for routing CPU and I/O-initiated transactions to target modules on the device.

Figure 7-1 shows a high level block diagram of the MPX coherency module (MCM).

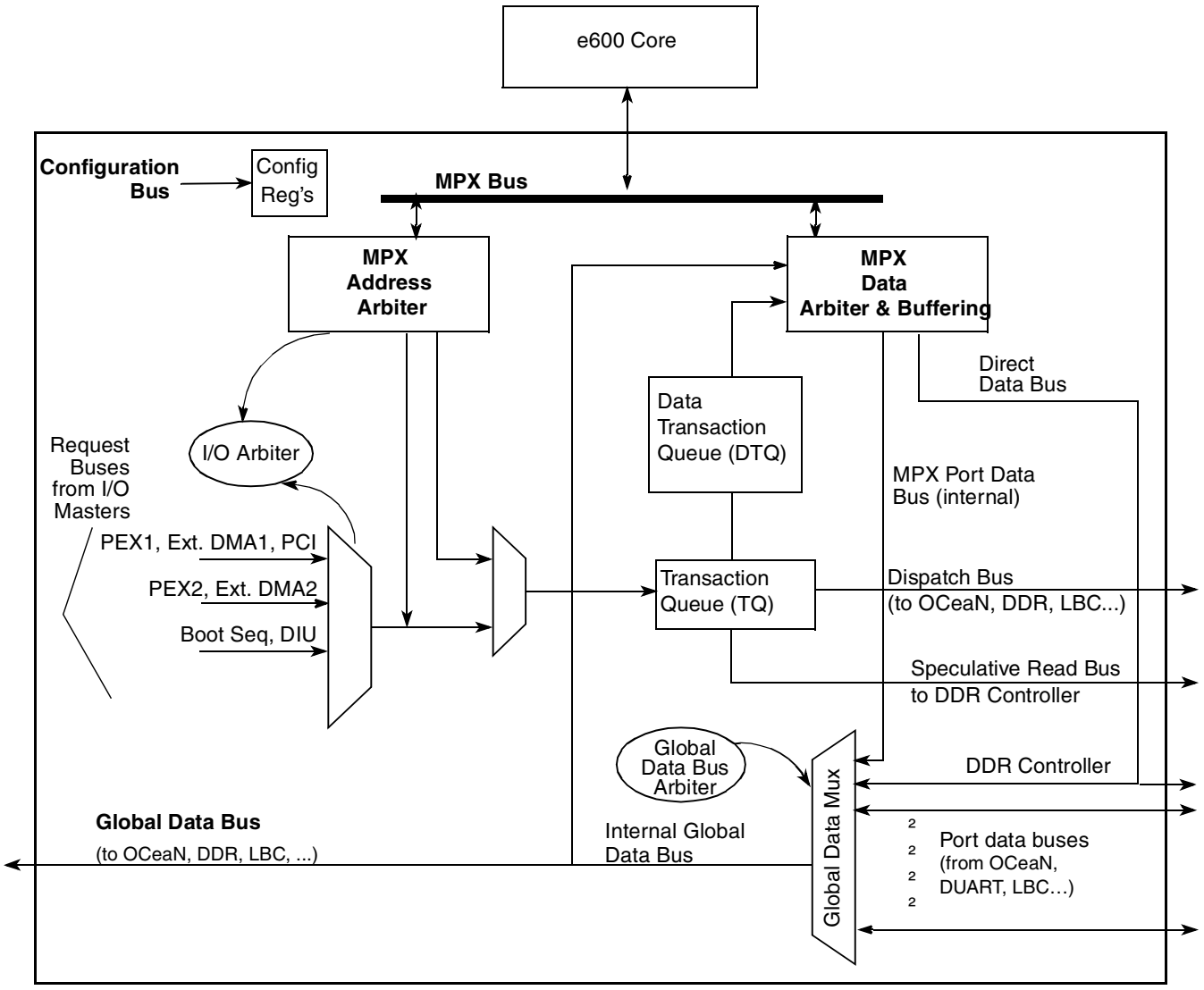


Figure 7-1. MPX Coherency Module (MCM) Overview

7.1.1 Overview

The MCM routes transactions initiated by the e600 core to the appropriate target interface on the device. In a manner analogous to a bridging router in a local area network, the MCM also forwards I/O-initiated transactions that are tagged with the global attribute (snoopable) onto the MPX bus interface. This allows the core caches to snoop these transactions as if they were locally initiated and to take actions to maintain coherency across cacheable memory.

7.2 Features

The MCM includes these distinctive features:

- Supports the MPX bus protocol of the core
  - Supports both address tenure streaming and data tenure streaming
  - Support for fully serialized mode on MPX bus
  - Split transaction support
  - Supports 36-bit addresses throughout the device, 64-bit MPX data bus, single beat, 2 beat bursts and 4 beat bursts of data transfer
  - Supports an 8-entry data transaction queue (DTQ)
  - Supports out-of-order data transactions on the MPX bus
- Provides full cache coherency between the core's caches and global I/O transactions
  - Supports data intervention on snoop hits
  - Supports window-of-opportunity and 'cache-to-cache' read intervention (snarfing)
- Supports 1 or 2 wait states from transfer start (TS) to address acknowledge (AACK) on the MPX bus. For a 3:1 clock ratio or greater core-to-platform frequency ratio there is a single wait state. For ratios less than 3:1 there are 2 wait states.
- Has 8 cache lines of data buffering for core-initiated read transactions and 8 cache lines of data buffering for MPX-initiated write transactions.
- Ensures ordering of I/O-initiated transactions on a per-master-ID basis
- Supports speculative read bus for low latency dispatch of reads from local prefetchable memory space (DDR controller).
- Provides low latency path for returning read data from local memory target to MPX data tenure.
- Error management. Errors can be programmed to generate interrupts to the e600 core, as described in the following sections:
  - Section 7.4.1.4, "Error Detect Register (EDR)"
  - Section 7.4.1.5, "Error Enable Register (EER)"
  - Section 7.4.1.6, "Error Attributes Capture Register (EATR)"
  - Section 7.4.1.7, "Error Low Address Register (ELADR)"
  - Section 7.4.1.8, "Error High Address Register (EHADR)"

7.3 Modes of Operation

There are various programmable options that are controlled by the MPX address bus configuration register (ABCR), the data bus configuration register (DBCR) and the port configuration register (PCR).

The address bus arbiter (ABA) uses a fair arbitration scheme to arbitrate for ownership of the address bus and the data bus. It can use either a least-recently-used (LRU) or round robin policy to arbitrate the address bus. The ABA also supports address tenure streaming and address bus parking. The data bus arbiter uses a round-robin policy to arbitrate the data bus.

7.4 Memory Map/Register Definition

Table 7-1 shows the memory mapped registers of the MCM 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 CCSRBAR together with the block base address and offset listed in Table 7-1. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 7-1. MCM Memory Map

Offset	Register	Access	Reset	Section/Page
MCM Registers—Block Base Address 0x0_1000				
0x000	MCM MPX address bus configuration register (ABCR)	R/W	0x0000_0003	7.4.1.1/7-5
0x008	MCM MPX data bus configuration register (DBCR)	R/W	0x0000_0003	7.4.1.2/7-6
0x010	MCM MPX port configuration register (PCR)	R/W	0x0*00_0000	7.4.1.3/7-6
0xBF8	MCM IP block revision register 1	R	0x0101_0200	—
0xBFC	MCM IP block revision register 2	R	0x0000_0002	—
0xE00	MCM error detect register (EDR)	w1c	All zeros	7.4.1.4/7-7
0xE08	MCM error enable register (EER)	R/W	All zeros	7.4.1.5/7-8
0xE0C	MCM error attributes capture register (EATR)	R	All zeros	7.4.1.6/7-9
0xE10	MCM error low address capture register (ELADR)	R	All zeros	7.4.1.7/7-10
0xE14	MCM error high address capture register (EHADR)	R	All zeros	7.4.1.8/7-10

In addition to those registers above, the MCMCR register controls the bandwidth and priority settings of the I/O masters with respect to the MCM, and it is described in Chapter 23, “Global Utilities.”

7.4.1 Register Descriptions

This section consists of detailed descriptions of the registers summarized in Table 7-1. Note that these registers are shown in big-endian format.

7.4.1.1 Address Bus Configuration Register (ABCR)

The MCM address bus configuration register (ABCR), shown in Figure 7-2, controls arbitration and streaming policies for the MPX bus.

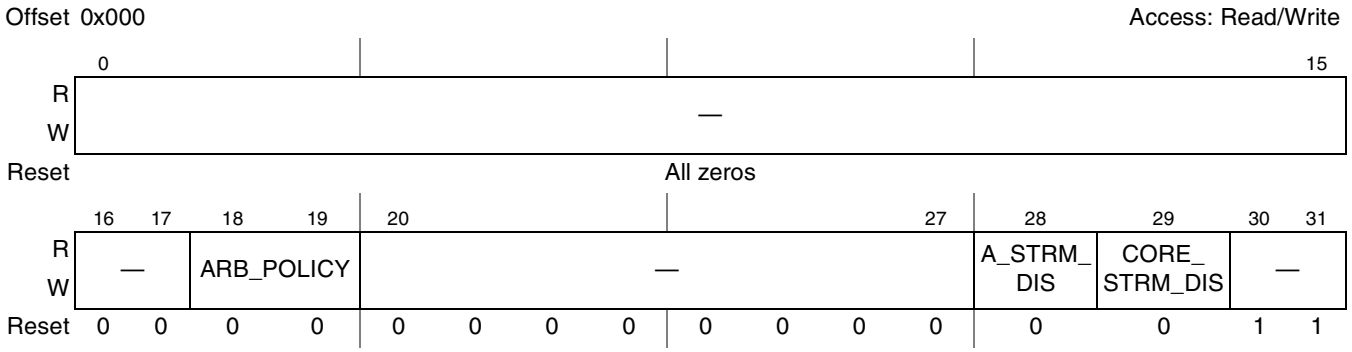


Figure 7-2. Address Bus Configuration Register (ABCR)

Table 7-2 describes the ABCR fields.

Table 7-2. ABCR Field Descriptions

Bits	Name	Description
0–17	—	Reserved
18–19	ARB_POLICY	Defines the address bus arbitration policy that is used on the MPX bus. This field may be used to tune for maximum performance for a particular application. 00 Longest-waiting (LW) 01 Round robin 10 Reserved 11 Reserved
20–27	—	Reserved
28	A_STRM_DIS	Controls whether the MCM, as a master of transactions, streams the address tenures. Note that this bit operates independently from the CORE_STRM_DIS bit. 0 MCM streams address tenures 1 MCM does not stream address tenures.
29	CORE_STRM_DIS	Controls whether the e600 core can stream commands onto the MPX bus. 0 Address tenures initiated by the e600 core are streamed. 1 Streaming of address tenures initiated by the e600 core not allowed.
30–31	—	Reserved

7.4.1.2 Data Bus Configuration Register (DBCR)

The data bus configuration register (DBCR), shown in Figure 7-3, controls the MPX data bus.

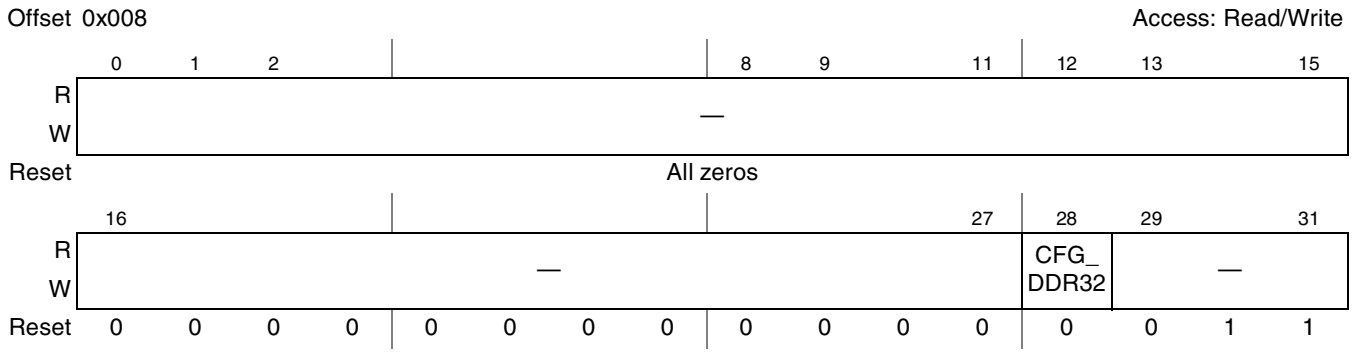


Figure 7-3. Data Bus Configuration Register (DBCR)

Table 7-3 describes the DBCR fields.

Table 7-3. DBCR Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28	CFG_DDR32	MCM configuration for 32-bit DDR controller operation. If the DDR controller is operated in 32-bit mode (DDR_SDRAM_CFG[32_BE] = 1), then this bit must be set to properly configure the MCM prior to enabling the DDR controller. 0 MCM is configured to operate with the DDR controller in 64-bit mode (DDR_SDRAM_CFG[32_BE] = 0). 1 MCM is configured to operate with the DDR controller in 32-bit mode (DDR_SDRAM_CFG[32_BE] = 1).
29–31	—	Reserved

7.4.1.3 Port Configuration Register (PCR)

The port configuration register (PCR), shown in Figure 7-4, enables the core's access to the MPX bus, and controls the priorities of core accesses with respect to the MCM.

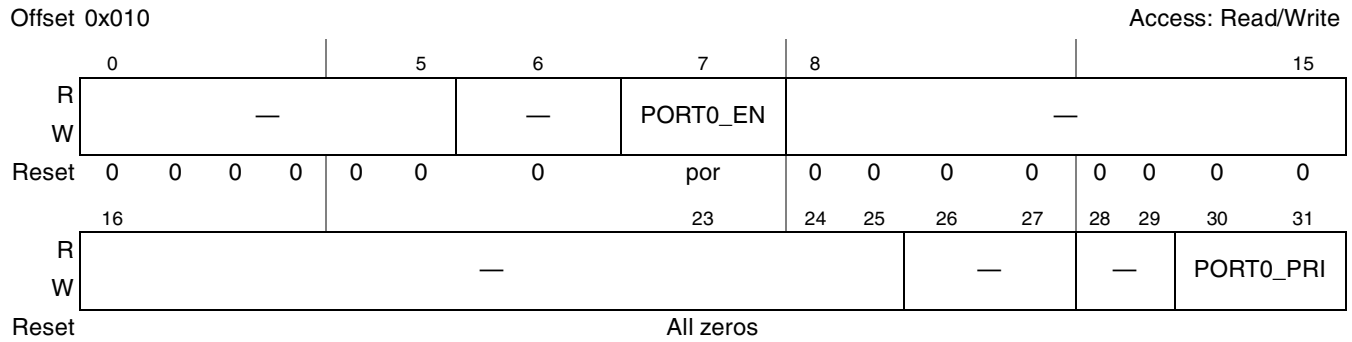


Figure 7-4. Port Configuration Register (PCR)

Table 7-4 describes the PCR fields.

Table 7-4. PCR Field Descriptions

Bits	Name	Description
0–6	—	Reserved
7	PORT0_EN	CPU port 0 enable. Identifies whether MPX port 0 (the core) can receive bus grants. The core boot configuration power-on reset pin (cfg_cpu_boot) determines the initial value of this bit. If the pin is sampled as a logic 1 at the negation of reset, core 0 is enabled to boot at the end of the POR sequence. Otherwise, the core cannot fetch its boot vector until an external host sets this PORT0_EN bit. 0 Boot holdoff mode. Port 0 (the core) arbitration is disabled on the MPX and no bus grants are issued. 1 Port 0 (the core is enabled and receives bus grants in response to bus requests. After this bit is set, it should not be cleared by software. It is not intended for port 0 to be dynamically enabled and disabled. This bit is only intended to end boot holdoff mode for core 0. See Section 4.4.3.6, "CPU Boot Configuration," for more information.
8–2527	—	Reserved
30–31	PORT0_PRI	Specifies the priority level of port 0 (the core). This priority level is used in streaming mode (when ABCR[A_STRM_DIS] is cleared) to determine whether an I/O master's MPX bus request can cause the MPX arbiter to terminate the core's streaming of address tenures. 00 Lowest priority level 01 Second lowest priority level 10 Highest priority level 11 Reserved

7.4.1.4 Error Detect Register (EDR)

The error detect register (EDR), shown in Figure 7-5, reports the detection of enabled errors.

Offset 0xE00 Access: w1c

	0	1	25	26	27	28	29	30	31
R	MULT_ERR	—				MHE	—		LAE
W	w1c					w1c			w1c
Reset	All zeros								

Figure 7-5. Error Detect Register (EDR)

Table 7-5 describes the EDR fields.

Table 7-5. EDR Field Descriptions

Bits	Name	Description
0	MULT_ERR	Multiple errors. Indicates the occurrence of multiple errors of the same type. Write 1 to clear. 0 Multiple errors of the same type were not detected. 1 Multiple errors of the same type were detected. Occurs when an error event is detected and its corresponding EDR bit is still set. When this occurs, the MCM signals an interrupt to the on-chip interrupt controller.
1–26	—	Reserved
26–27	—	Reserved

Table 7-5. EDR Field Descriptions (continued)

Bits	Name	Description
28	MHE	Multiple hit error. The MCM always detects multiple hit errors. This error occurs when a core issues a snoop transaction (and the same master signals hit for the transaction that it is mastering), and that transaction also maps to a local MCM target; it is also considered a multiple hit error event. When this error occurs and none of the MCM error capture registers are set, the MCM captures the address and attributes of the transaction into the EATR, ELADR, and EHADR registers. Furthermore, if EER[MHEE] is set while EDR[MHE] is set, the MCM signals an interrupt to the on-chip interrupt controller. This bit is cleared (if it is set) by writing a logic 1 to it. 0 Multiple hit error was not detected on the MPX bus. 1 Multiple hit error was detected on the MPX bus.
29–30	—	Reserved
31	LAE	Local access error. Write 1 to clear. The following cases can generate LAEs in the MCM: • Non address-only transaction does not map to any target. In this case the MCM injects read responses (with the corrupt attribute set) and write data is dropped. • Source and target IDs indicate that an OCN port initiated a transaction that targets an OCN port. This loopback behavior can result from programming errors where inbound ATMU window targets are inconsistent with targets configured in the local access windows for a given address range. For this type of LAE, the dispatch (to the OCN target in this case) is not blocked; the LAE error is reported, but the transaction is still sent to its OCN target. When this type of error occurs, and none of the MCM error capture registers are set, the MCM captures the address and attributes of the transaction into the EATR, ELADR, and EHADR registers. Furthermore, if EER[LAEE] is set while EDR[LAE] is set, the MCM signals an interrupt to the on-chip interrupt controller. This bit is cleared (if it is set) by writing a logic 1 to it. 0 Local access error has not occurred. 1 Local access error occurred.

7.4.1.5 Error Enable Register (EER)

The error enable register (EER), shown in Figure 7-6, enables the reporting of MCM-detected errors as interrupts (through the on-chip interrupt controller).

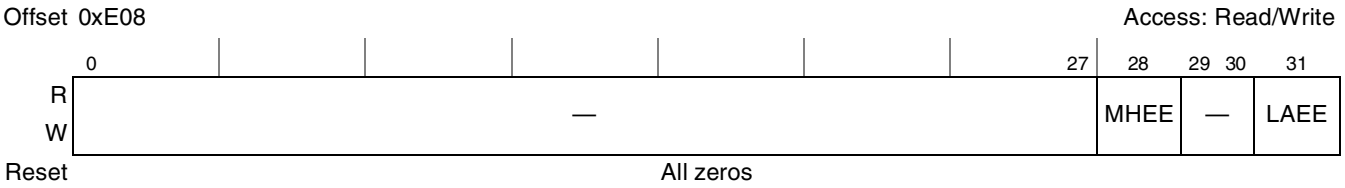


Figure 7-6. Error Enable Register (EER)

Table 7-6 describes the EER fields.

Table 7-6. EER Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28	MHEE	Multiple hit error enable. Enables the reporting of an interrupt in the case of a multiple hit error. If this bit is set and a multiple hit error occurs (setting EDR[MHE]), the MCM signals an interrupt to the on-chip interrupt controller. For more information on multiple hit errors, see the description for EDR[MHE]. 0 Disables reporting of multiple hit errors as interrupts. 1 Enables reporting of multiple hit errors as interrupts.

Table 7-6. EER Field Descriptions (continued)

Bits	Name	Description
29–30	—	Reserved
31	LAEE	Local access error enable. Enables the reporting of an interrupt in the case of a local access error. If this bit is set and a local access error occurs (setting EDR[LAE]), the MCM signals an interrupt to the on-chip interrupt controller. For more information on local access errors, see the description for EDR[LAE]. 0 Disables reporting of local access errors as interrupts. 1 Enables reporting of local access errors as interrupts.

7.4.1.6 Error Attributes Capture Register (EATR)

The error attributes capture register (EATR), shown in Figure 7-7, captures relevant information about the transaction that causes an error detected by the MCM.

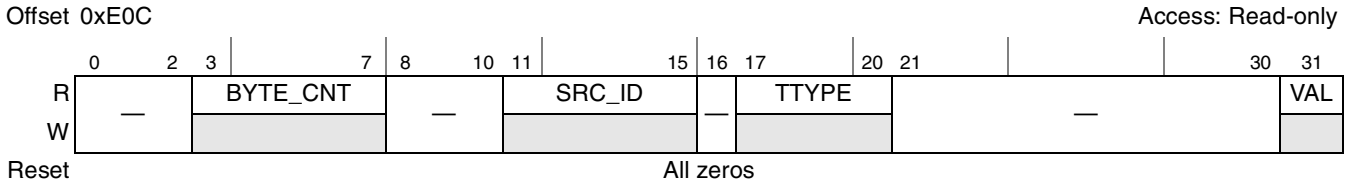


Figure 7-7. Error Attributes Capture Register (EATR)

Table 7-7 describes the EATR fields.

Table 7-7. EATR Field Descriptions

Bits	Name	Description
0–2	—	Reserved
3–7	BYTE_CNT	Byte count. Specifies the transaction byte count. All other values (not shown) are reserved. 00000 32 bytes 00001 1 byte 00010 2 bytes 00011 3 bytes 00100 4 bytes 00101 5 bytes 00110 6 bytes 00111 7 bytes 01000 8 bytes 10000 16 bytes
8–10	—	Reserved
11–15	SRC_ID	Source ID. Specifies the source device mastering the transaction. 00000 PCI Interface 00001 PCI Express interface 2 00010 PCI Express interface 1 00011–01000 Reserved 01001 DIU 01010 Boot sequencer 01011–01111 Reserved 10000 Core (instruction) 10001 Core (data) 10010–10100 Reserved 10101 DMA 1 10110 DMA 2 10111–11111 Reserved
16	—	Reserved

Table 7-7. EATR Field Descriptions (continued)

Bits	Name	Description
17–20	TTYPE	Transaction type. Defined as follows: 0000 Write 0001 Reserved 0010 Write with allocate 0011 Write with allocate with lock 0100 Address only transaction (other than clean or flush) 0101–0111 Reserved 1000 Read 1001 Read with unlock 1010 Clean or flush 1011 Reserved 1100 Read with clear atomic 1101 Read with set atomic 1110 Read with decrement atomic 1111 Read with increment atomic
21–30	—	Reserved
31	VAL	Register data valid. 0 EATR does not contain valid information. 1 EATR contains valid information.

7.4.1.7 Error Low Address Register (ELADR)

The error low address register (ELADR), shown in Figure 7-8, captures the lowest 32 bits of the 36-bit physical address of the transaction. The value in this register should be qualified with the EATR[VAL] bit.

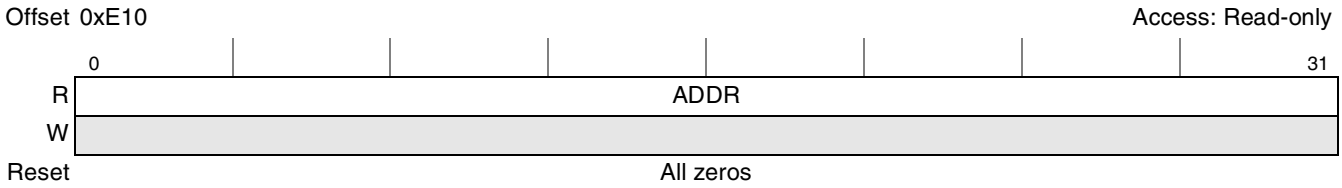


Figure 7-8. Error Low Address Register (ELADR)

Table 7-8 describes the ELADR fields.

Table 7-8. ELADR Field Descriptions

Bits	Name	Description
0–31	ADDR	Address. Specifies the lower-order 32 bits of the 36-bit address of the transaction. Qualified by EATR[VAL].

7.4.1.8 Error High Address Register (EHADR)

The error high address register (EHADR), shown in Figure 7-9, captures the highest 4 bits of the 36-bit physical address of the transaction. The value in this register should be qualified with EATR[VAL].

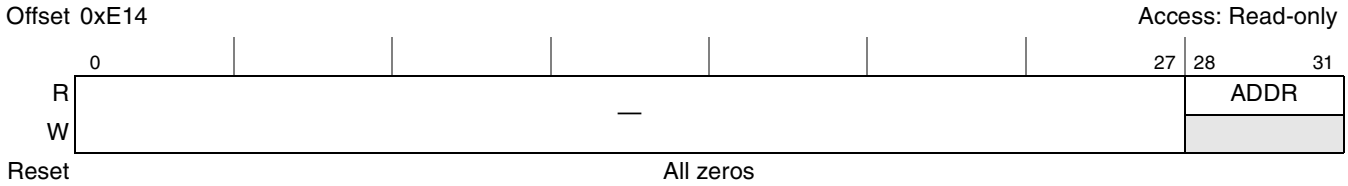


Figure 7-9. Error High Address Register (EHADR)

Table 7-9 describes EHADR fields.

Table 7-9. EHADR Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28–31	ADDR	Address. Specifies the high-order 4 bits of the 36-bit address of the transaction. Qualified by EATR[VAL].

7.5 Functional Description

The following is a general description of MCM operation.

7.5.1 I/O Arbiter

Figure 7-1 shows the I/O arbiter block that manages I/O-initiated address tenure requests arriving on the request buses. The request buses compete for access to the MCM, which can only process one request at a time. The MCM uses two factors to select the winning request bus: the primary factor is requested bandwidth and the secondary factor is longest waiting/least recently granted status. By default, all requesters start requesting low levels of bandwidth. A starvation avoidance algorithm ensures that low bandwidth requesters make forward progress in the presence of high bandwidth requesters.The transaction from the winning request bus competes with core requests for the MPX bus and entry into the transaction queue.

7.5.2 MPX Address Arbiter

Figure 7-1 shows the MPX address arbiter block which coordinates the entry of new transactions into the MCM’s transaction queue. It handles arbitration for requests to use the MPX address bus from the core and the winning request bus (for I/O-initiated snoopable transactions) and consequently controls when these new transactions can enter the transaction queue.

Because the MPX bus operates most efficiently when it streams transactions from one initiator, the MPX address arbiter alternates grants between streams of transactions from the core and from the winner of the I/O arbiter. The arbiter uses the priority of the requests to limit streaming. If the priority of a new request is higher than that of a stream in progress, then the higher priority transaction interrupts the stream in progress. The priority of core transactions is set by the PORT*n*_PRI field in PCR.

7.5.3 Transaction Queue

The MCM's transaction queue performs three basic functions: target mapping and dispatching, enforcement of ordering, and enforcement of coherency. The address of each transaction is compared with the address range defined for each local access window, and the transaction is then routed to the appropriate target interface. Even though the MCM allows for pipelining of transactions (and treats address tenures independently from data tenures), the address tenures of all transactions issued from I/O masters (masters other than the core) are dispatched to their target interfaces in the same order they are submitted. For those I/O masters requiring higher levels of performance, ordering is only enforced for a given master as is needed, so as not to degrade performance unnecessarily. For those transactions accessing

address space marked as snoopable, or space that may be cached by the core, the MCM enforces coherency, mirroring those transactions on the MPX bus for snooping purposes, and taking castouts or interventions from the cores as necessary.

7.5.4 Global Data Multiplexor

Figure 7-1 shows how the global data multiplexor takes data bus connections and multiplexes them onto one 128-bit wide global data bus. The global data multiplexor allows initiators of write transactions to route data to their targets and read targets to return data to the initiators.

7.5.4.1 Direct Data Bus

In order to further optimize the performance of CPU-to-DDR memory transactions, the MCM has a dedicated direct data bus from the DDR controller to the MPX bus. Thus, core accesses to the DDR controller can complete in fewer cycles, shortcutting around the global data multiplexing logic.

7.5.5 MPX Interface

Figure 7-1 shows the MPX interface for both MPX address and data tenures. This interface formats MPX address tenures for the MCM transaction queue. It also contains the queueing and buffering needed to manage outstanding MPX data tenures. The buffers receive core-initiated write and I/O-initiated read data from the core write (128-bit wide) and read (128-bit wide) data buses and route it through the global data multiplexor to the global data bus. The buffers also receive core-initiated read and I/O-initiated write data from the global data bus and forward them onto the MPX data bus (64 bits).

7.6 Initialization/Application Information

If the core is to be used to initialize the device, the corresponding CPU boot configuration power-on reset pin () should be pulled high to initially set the PCR[PORT0_EN] bit. See Chapter 4, “Reset, Clocking, and Initialization,” for more information on power-up reset initialization.

If any device other than the core, such as the boot sequencer or a PCI Express device, is used to initialize the device, the CPU boot configuration power-on reset pin should be pulled low to initially clear PCR[PORT0_EN]. This prevents the core from accessing any configuration registers or local memory space during initialization. However, in any such system, one step near the end of the initialization routine must set PCR[PORT0_EN] to enable the core.

PCR[PORT0_PRI]specifies the priority level associated with all core-initiated transactions. This value allows users running time-critical applications to adjust the average response latency of transactions initiated by the core compared to those initiated by I/O masters. These priority levels affect whether the core’s requests can interrupt the streaming of address tenures initiated by the MCM on behalf of I/O masters. Only transactions with a priority greater than the current MPX transaction can interrupt streaming. The higher the core’s priority, the lower the average latency needed for it to obtain bus grants from the MCM, because it can interrupt lower priority streaming. The default values of zero give all core-initiated transactions the lowest priority, which prevents the cores from interrupting I/O master transaction streams.

Chapter 8
DDR Memory Controller

8.1 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 8-1 is a high-level block diagram of the DDR memory controller with its associated interfaces. Section 8.5, "Functional Description," contains detailed figures of the controller.

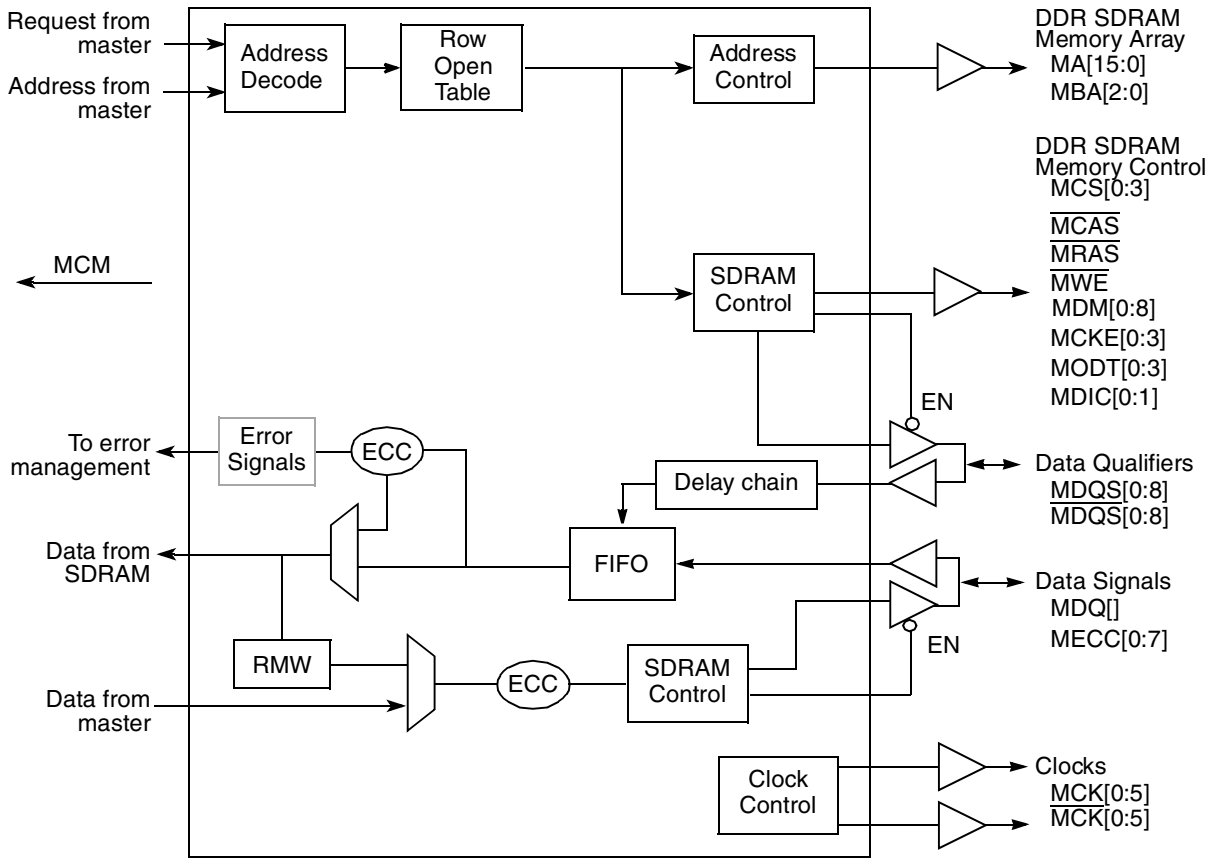


Figure 8-1. DDR Memory Controller Simplified Block Diagram

8.2 Features

The DDR memory controller includes these distinctive features:

- Support for DDR2 and DDR SDRAM
- 32-/40-bit SDRAM for DDR and DDR2 and
- Programmable settings for meeting all SDRAM timing parameters
- Support for the following SDRAM configurations:
  - — As many as four physical banks (chip selects), each bank independently addressable
    - – 64-Mbit to 4-Gbit devices depending on internal device configuration with ×8/×16/×32 data ports (no direct ×4 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

8.2.1 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].

8.3 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.

8.3.1 Signals Overview

Memory controller signals are grouped as follows:

- Memory interface signals
- Clock signals
- Debug signals

Table 8-1 shows how DDR memory controller external signals are grouped. The device hardware specification has a pinout diagram showing pin numbers. It also lists all electrical and mechanical specifications.

Table 8-1. DDR Memory Interface Signal Summary

Name	Function/Description	Reset	Pins	I/O
MDQ[]	Data bus	All zeros		I/O
MDQS[]	Data strobes	All zeros		I/O
$\overline{\text{MDQS}}$[]	Complement data strobes	All ones		I/O

Table 8-1. DDR Memory Interface Signal Summary (continued)

Name	Function/Description	Reset	Pins	I/O
MECC[0:7]	Error checking and correcting	All zeros	8	I/O
$\overline{\text{MCAS}}$	Column address strobe	One	1	O
MA[15:0]	Address bus	All zeros	16	O
MBA[2:0]	Logical bank address	All zeros	3	O
$\overline{\text{MCS}}$[0:3]	Chip selects	All ones	4	O
$\overline{\text{MWE}}$	Write enable	One	1	O
$\overline{\text{MRAS}}$	Row address strobe	One	1	O
MDM[]	Data mask	All zeros		O
MCK[0:5]	DRAM clock outputs	All zeros	6	O
$\overline{\text{MCK}}$[0:5]	DRAM clock outputs (complement)	All zeros	6	O
MCKE[0:3]	DRAM clock enable	All zeros	4	O
MODT[0:3]	DRAM on-die termination external control.	All zeros	4	O
MDVAL	Memory debug data valid	Zero	1	O
MSRCID[0:4]	Memory debug source ID	All zeros	5	O
MDIC[0:1]	Driver impedance calibration		2	I/O

Table 8-2 shows the memory address signal mappings.

Table 8-2. Memory Address Signal Mappings

Signal Name (Outputs)		JEDEC DDR DIMM Signals (Inputs)
msb	MA15	A15
	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

8.3.2 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.

8.3.2.1 Memory Interface Signals

Table 8-3 describes the DDR controller memory interface signals.

Table 8-3. Memory Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
MDQ[]	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.
MECC[0:7]	I/O	Error checking and correcting codes. Input and output signals 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 8.5.11, "Error Checking and Correcting (ECC)."	
		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

Table 8-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
MA[15:0]	O	Address bus. Memory controller outputs for the address to the DRAM. MA[15:0] carry 16 of the address bits for the DDR memory interface corresponding to the row and column address bits. MA0 is 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 8-47Table 8-2 for a complete description of the mapping of these signals.
		Timing	Assertion/Negation—The address lines are only driven when the controller has a command scheduled to issue on the address/CMD bus; otherwise they will be at high-Z. 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
MBA[2:0]	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 must 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 8-47Table 8-2 describes the mapping of these signals in all cases.
		Timing	Assertion/Negation—Same timing as MA*n* High impedance—Same timing as MA*n*
$\overline{\text{MCAS}}$	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 8-53 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 8.4.1.4, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)," Section 8.4.1.5, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)," Section 8.4.1.6, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)," and Section 8.4.1.3, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)." High impedance—$\overline{\text{MCAS}}$ is always driven unless the memory controller is disabled.
$\overline{\text{MRAS}}$	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 8-53 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 8.4.1.4, "DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)," Section 8.4.1.5, "DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)," Section 8.4.1.6, "DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)," and Section 8.4.1.3, "DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)." High impedance—$\overline{\text{MRAS}}$ is always driven unless the memory controller is disabled.

Table 8-3. Memory Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
$\overline{\text{MCS}}$[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 8.4.1.1, "Chip Select Memory Bounds (CSn_BNDS)," and Section 8.4.1.2, "Chip Select Configuration (CSn_CONFIG)." The DDR controller asserts one of the $\overline{\text{MCS}}$[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.
$\overline{\text{MWE}}$	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 8-53 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.
MODT[0:3]	O	On-Die termination. Memory controller outputs for the ODT to the DRAM. MODT[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.
MDIC[0:1]	I/O	Driver impedance calibration. Note that the MDIC signals require the use of 18.2-Ω precision 1% resistors; MDIC0 must be pulled to GND, while MDIC1 must be pulled to GV_{DD}. Section 8.4.1.20, "DDR Control Driver Register 2 (DDRCDR_2)," for more information on these signals.	
		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.

8.3.2.2 Clock Interface Signals

Table 8-4 contains the detailed descriptions of the clock signals of the DDR controller.

Table 8-4. 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 8.5.4.1, "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.

Table 8-4. Clock Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
MCKE[0:3]	O	Clock enable. Output signals used as the clock enables to the SDRAM. MCKE[0:3] 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[0:3] 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.

8.3.2.3 Debug Signals

The debug signals MSRCID[0:4] and MDVAL have no function in normal DDR controller operation. A detailed description of these signals can be found in Section 25.4.2, "DDR SDRAM Interface Debug."

8.4 Memory Map/Register Definition

Table 8-5 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 8-5. 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	All zeros	8.4.1.1/8-11
0x008	CS1_BNDS—Chip select 1 memory bounds	R/W	All zeros	8.4.1.1/8-11
0x010	CS2_BNDS—Chip select 2 memory bounds	R/W	All zeros	8.4.1.1/8-11
0x018	CS3_BNDS—Chip select 3 memory bounds	R/W	All zeros	8.4.1.1/8-11
0x080	CS0_CONFIG—Chip select 0 configuration	R/W	All zeros	8.4.1.2/8-11
0x084	CS1_CONFIG—Chip select 1 configuration	R/W	All zeros	8.4.1.2/8-11
0x088	CS2_CONFIG—Chip select 2 configuration	R/W	All zeros	8.4.1.2/8-11
0x08C	CS3_CONFIG—Chip select 3 configuration	R/W	All zeros	8.4.1.2/8-11

Table 8-5. DDR Memory Controller Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	All zeros	8.4.1.3/8-13
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	8.4.1.4/8-14
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	All zeros	8.4.1.5/8-16
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	All zeros	8.4.1.6/8-18
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	8.4.1.7/8-20
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	All zeros	8.4.1.8/8-23
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	All zeros	8.4.1.9/8-24
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	All zeros	8.4.1.10/8-25
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	All zeros	8.4.1.11/8-26
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	All zeros	8.4.1.12/8-28
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	All zeros	8.4.1.13/8-29
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	8.4.1.14/8-29
0x140–0x144	Reserved	—	—	—
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	All zeros	8.4.1.15/8-30
0x14C	DDR_INIT_EXT_ADDR—DDR training initialization extended address	R/W	All zeros	8.4.1.16/8-30
0x150–0xB1F	Reserved	—	—	—
0xB20	DDRDSR_1—DDR Debug Status Register 1	R	All zeros	8.4.1.17/8-31
0xB24	DDRDSR_2—DDR Debug Status Register 2	R	All zeros	8.4.1.18/8-32
0xB28	DDRCDR_1—DDR Control Driver Register 1	R/W	All zeros	8.4.1.19/8-32
0xB2C	DDRCDR_2—DDR Control Driver Register 2	R/W	All zeros	8.4.1.20/8-34
0xB30–0xBF7	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0x*nnnn_nnnn*[1]	8.4.1.21/8-34
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x0000_0001	8.4.1.22/8-35
0xE00–0xE58	Reserved	—	—	—
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	All zeros	8.4.1.23/8-35
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	All zeros	8.4.1.24/8-36
0xE08	ERR_INJECT—Memory data path error injection mask ECC	R/W	All zeros	8.4.1.25/8-36
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	All zeros	8.4.1.26/8-37
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	All zeros	8.4.1.27/8-37
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	All zeros	8.4.1.28/8-38
0xE40	ERR_DETECT—Memory error detect	w1c	All zeros	8.4.1.29/8-38
0xE44	ERR_DISABLE—Memory error disable	R/W	All zeros	8.4.1.30/8-39
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	All zeros	8.4.1.31/8-40
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	All zeros	8.4.1.32/8-41
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	All zeros	8.4.1.33/8-42

Table 8-5. DDR Memory Controller Memory Map (continued)

Offset	Register	Access	Reset	Section/Page
0xE54	CAPTURE_EXT_ADDRESS—Memory error extended address capture	R/W	All zeros	8.4.1.34/8-42
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	All zeros	8.4.1.35/8-43

1 Implementation-dependent reset values are listed in specified section/page.

8.4.1 Register Descriptions

This section describes the DDR memory controller registers. Shading indicates reserved fields that should not be written.

8.4.1.1 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 8-2.

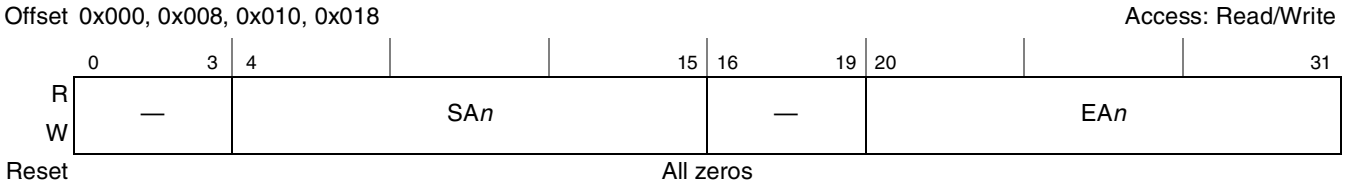


Figure 8-2. Chip Select Bounds Registers (CS*n*_BNDS)

Table 8-6 describes the CS*n*_BNDS register fields.

Table 8-6. CS*n*_BNDS Field Descriptions

Bits	Name	Description
0–3	—	Reserved
4–15	SA*n*	Starting address for chip select (bank) *n*. This value is compared against the 12 msbs of the 36-bit address.
16–19	—	Reserved
20–31	EA*n*	Ending address for chip select (bank) *n*. This value is compared against the 12 msbs of the 36-bit address.

8.4.1.2 Chip Select Configuration (CS*n*_CONFIG)

The chip select configuration (CS*n*_CONFIG) registers shown in Figure 8-3 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.

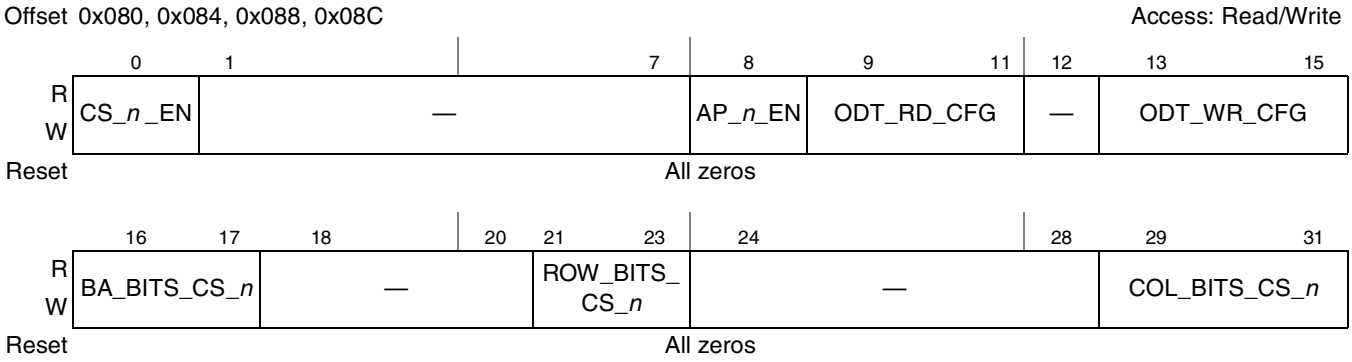


Figure 8-3. Chip Select Configuration Register (CS*n*_CONFIG)

Table 8-7 describes the CS*n*_CONFIG register fields.

Table 8-7. 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 or 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 101–111Reserved
12	—	Reserved

Table 8-7. 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 or 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 101–111Reserved
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 8-49. 00 2 logical bank bits 01 3 logical bank bits 10–11Reserved
18–20	—	Reserved
21–23	ROW_BITS_CS_*n*	Number of row bits for SDRAM on chip select *n*. See Table 8-49 for details. 000 12 row bits 001 13 row bits 010 14 row bits 011 15 row bits 100 16 row bits 101–111 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

8.4.1.3 DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

DDR SDRAM timing configuration register 3, shown in Figure 8-4, sets the extended refresh recovery time, which is combined with TIMING_CFG_1[REFREC] to determine the full refresh recovery time.

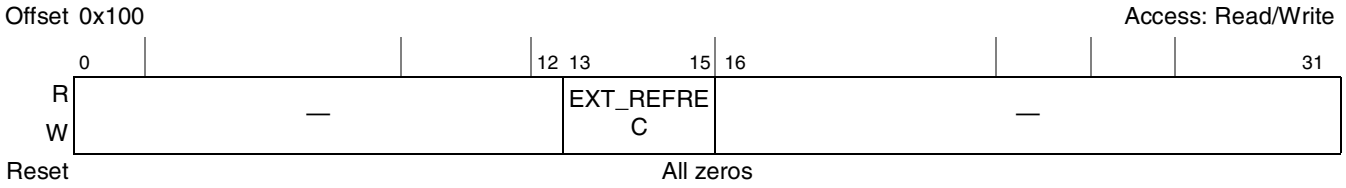


Figure 8-4. DDR SDRAM Timing Configuration 3 (TIMING_CFG_3)

Table 8-8 describes TIMING_CFG_3 fields.

Table 8-8. TIMING_CFG_3 Field Descriptions

Bits	Name	Description
0–12	—	Reserved
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

8.4.1.4 DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)

DDR SDRAM timing configuration register 0, shown in Figure 8-5, sets the number of clock cycles between various SDRAM control commands.

Offset 0x104 Access: Read/Write

	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 1	0 0 0 0	0 0 0 1	0 0 0 0	0 1 0 1

Figure 8-5. DDR SDRAM Timing Configuration 0 (TIMING_CFG_0)

Table 8-9 describes TIMING_CFG_0 fields.

Table 8-9. 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 8-9. 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
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
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
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

Table 8-9. TIMING_CFG_0 Field Descriptions (continued)

Bits	Name	Description
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

8.4.1.5 DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)

DDR SDRAM timing configuration register 1, shown in Figure 8-6, 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 8-6. DDR SDRAM Timing Configuration 1 (TIMING_CFG_1)

Table 8-10 describes TIMING_CFG_1 fields.

Table 8-10. TIMING_CFG_1 Field Descriptions

Bits	Name	Description
0	—	Reserved
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

Table 8-10. TIMING_CFG_1 Field Descriptions (continued)

Bits	Name	Description
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 8.4.1.8, "DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2).") 0000 Reserved — 1000 4.5 clocks 0001 1 clock — 1001 5 clocks 0010 1.5 clocks — 1010 5.5 clocks 0011 2 clocks — 1011 6 clocks 0100 2.5 clocks — 1100 6.5 clocks 0101 3 clocks — 1101 7 clocks 0110 3.5 clocks — 1110 7.5 clocks 0111 4 clocks — 1111 8 clocks
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 — 0011 11 clocks 0001 9 clocks — ... 0010 10 clocks — 1111 23 clocks
20	—	Reserved
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

Table 8-10. TIMING_CFG_1 Field Descriptions (continued)

Bits	Name	Description
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
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

8.4.1.6 DDR SDRAM Timing Configuration 2 (TIMING_CFG_2)

DDR SDRAM timing configuration 2, shown in Figure 8-7, sets the clock delay to data for writes.

Offset 0x10C — Access: Read/Write

	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 8-7. DDR SDRAM Timing Configuration 2 Register (TIMING_CFG_2)

Table 8-11 describes the TIMING_CFG_2 fields.

Table 8-11. 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

Table 8-11. TIMING_CFG_2 Field Descriptions (continued)

Bits	Name	Description
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. 00000READ_LAT + 1 00001Reserved 00010READ_LAT 00011READ_LAT + 1/4 00100READ_LAT + 1/2 00101READ_LAT + 3/4 00110READ_LAT + 1 00111READ_LAT + 5/4 01000READ_LAT + 3/2 01001READ_LAT + 7/4 01010READ_LAT + 2 01011READ_LAT + 9/4 01100READ_LAT + 5/2 01101READ_LAT + 11/4 01110READ_LAT + 3 01111READ_LAT + 13/4 10000READ_LAT + 7/2 10001READ_LAT + 15/4 10010READ_LAT + 4 10011READ_LAT + 17/4 10100READ_LAT + 9/2 10101READ_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
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 8.5.7, “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

Table 8-11. TIMING_CFG_2 Field Descriptions (continued)

Bits	Name	Description
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. 000000Reserved 0000011 cycle 0000102 cycles 0000113 cycles 0001004 cycles ... 01001119 cycles 01010020 cycles 010101–111111 Reserved

1 For CPO decodings other than 00000 and 11111, 'READ_LAT' is rounded up to the next integer value.

8.4.1.7 DDR SDRAM Control Configuration (DDR_SDRAM_CFG)

The DDR SDRAM control configuration register, shown in Figure 8-8, 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

	0	1	2	3	4	5	6	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

	16	17	18	19	20	21	22	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 8-8. DDR SDRAM Control Configuration Register (DDR_SDRAM_CFG)

Table 8-12 describes the DDR_SDRAM_CFG fields.

Table 8-12. 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.

Table 8-12. DDR_SDRAM_CFG Field Descriptions (continued)

Bits	Name	Description
2	ECC_EN	ECC enable. Note that non-correctable 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. 000–001Reserved 010 DDR1 SDRAM 011 DDR2 SDRAM 100 Reserved 101 Reserved 110 Reserved 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 1 32-bit bus is used.If the DDR controller is configured to operate in 32-bit mode (32_BE = 1), then DBCR[CFG_DDR32] in the MCM configuration registers must be set to properly configure the MCM prior to enabling the DDR controller.
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;
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 is 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.

Table 8-12. DDR_SDRAM_CFG Field Descriptions (continued)

Bits	Name	Description
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.) 0000000No external memory banks are interleaved 1000000External memory banks 0 and 1 are interleaved 0100000External memory banks 2 and 3 are interleaved 1100000External memory banks 0 and 1 are interleaved together **and** banks 2 and 3 are interleaved together *xx*00100External 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 8.4.1.19, "DDR Control Driver Register 1 (DDRCDR_1)." 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
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 8.4.1.15, "DDR Initialization Address (DDR_INIT_ADDR)," for details on avoiding ECC errors in this mode.

8.4.1.8 DDR SDRAM Control Configuration 2 (DDR_SDRAM_CFG_2)

The DDR SDRAM control configuration register 2, shown in Figure 8-9, provides more control configuration for the DDR controller.

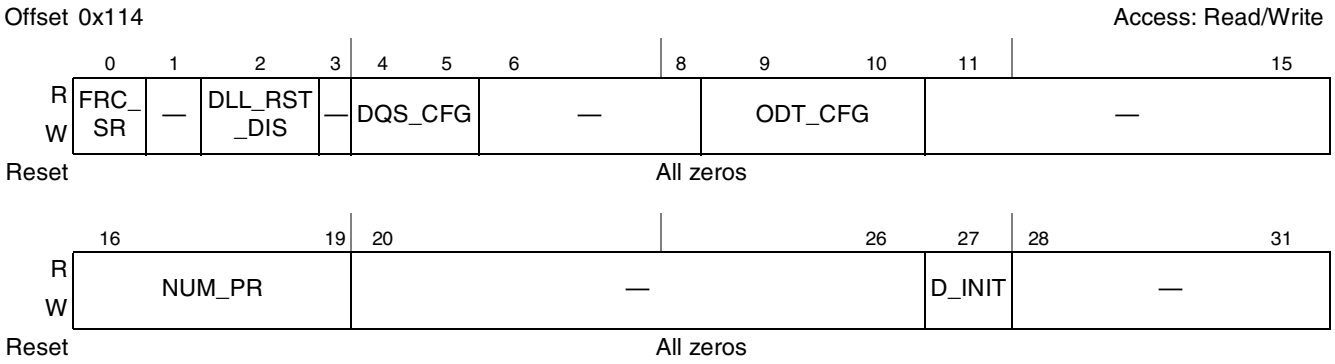


Figure 8-9. DDR SDRAM Control Configuration Register 2 (DDR_SDRAM_CFG_2)

Table 8-13 describes the DDR_SDRAM_CFG_2 fields.

Table 8-13. 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	—	Reserved. Should be cleared.
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 Differential DQS signals are used for DDR2 support. 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 8.4.1.19, “DDR Control Driver Register 1 (DDRCDR_1),” 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.

Table 8-13. DDR_SDRAM_CFG_2 Field Descriptions (continued)

Bits	Name	Description
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
20–24	—	Reserved
20–26	—	Reserved
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

8.4.1.9 DDR SDRAM Mode Configuration (DDR_SDRAM_MODE)

The DDR SDRAM mode configuration register, shown in Figure 8-10, sets the values loaded into the DDR's mode registers.

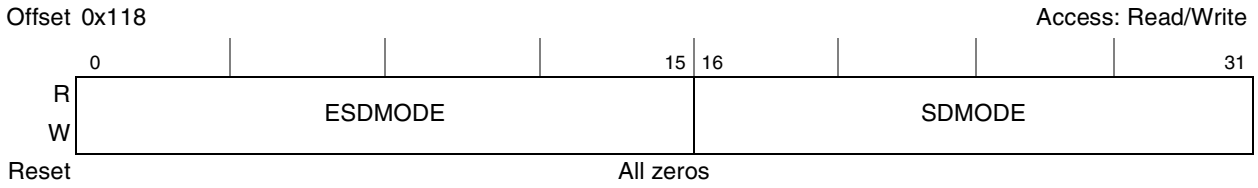


Figure 8-10. DDR SDRAM Mode Configuration Register (DDR_SDRAM_MODE)

Table 8-14 describes the DDR_SDRAM_MODE fields.

Table 8-14. 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 8-10, 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 8-10, 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].

8.4.1.10 DDR SDRAM Mode 2 Configuration (DDR_SDRAM_MODE_2)

The DDR SDRAM mode 2 configuration register, shown in Figure 8-11, sets the values loaded into the DDR's extended mode 2 and 3 registers (for DDR2).

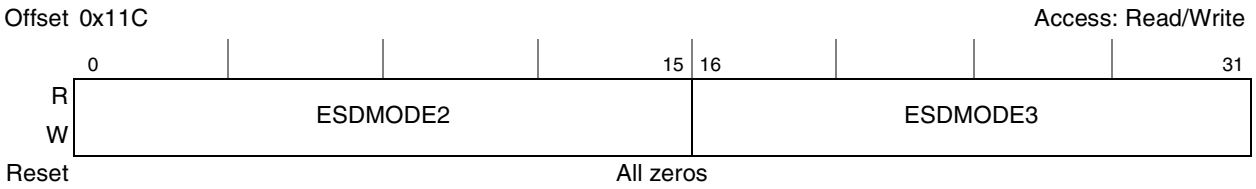


Figure 8-11. DDR SDRAM Mode 2 Configuration Register (DDR_SDRAM_MODE_2)

Table 8-15 describes the DDR_SDRAM_MODE_2 fields.

Table 8-15. 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 8-11, 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 8-11, corresponds to ESDMODE3[15]. The msb of the SDRAM extended mode 3 register value must be stored at ESDMODE3[0].

8.4.1.11 DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

The DDR SDRAM mode control register, shown in Figure 8-12, 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

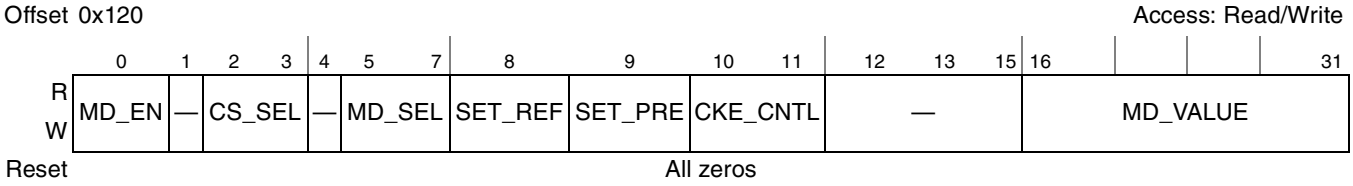


Figure 8-12. DDR SDRAM Mode Control Register (DDR_SDRAM_MD_CNTL)

Table 8-16 describes the fields of this register. Table 8-17 shows the user how to set the fields of this register to accomplish the above tasks.

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 8-16. 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

Table 8-16. DDR_SDRAM_MD_CNTL Field Descriptions (continued)

Bits	Name	Description
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.
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 8-17 shows how DDR_SDRAM_MD_CNTL fields should be set for each of the tasks described above.

Table 8-17. 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 8-16.	—	Selects logical bank	—
MD_VALUE	Value written to mode register	—	Only bit five is significant. See Table 8-16.	—
CKE_CNTL	0	0	0	See Table 8-16.

8.4.1.12 DDR SDRAM Interval Configuration (DDR_SDRAM_INTERVAL)

The DDR SDRAM interval configuration register, shown in Figure 8-13, 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.

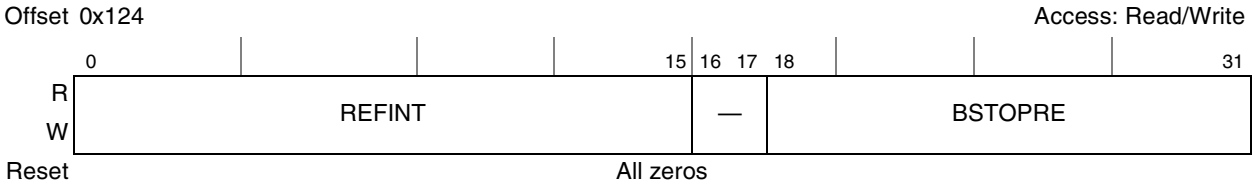


Figure 8-13. DDR SDRAM Interval Configuration Register (DDR_SDRAM_INTERVAL)

Table 8-18 describes the DDR_SDRAM_INTERVAL fields.

Table 8-18. 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.

8.4.1.13 DDR SDRAM Data Initialization (DDR_DATA_INIT)

The DDR SDRAM data initialization register, shown in Figure 8-14, provides the value that is used to initialize memory if DDR_SDRAM_CFG2[D_INIT] is set.

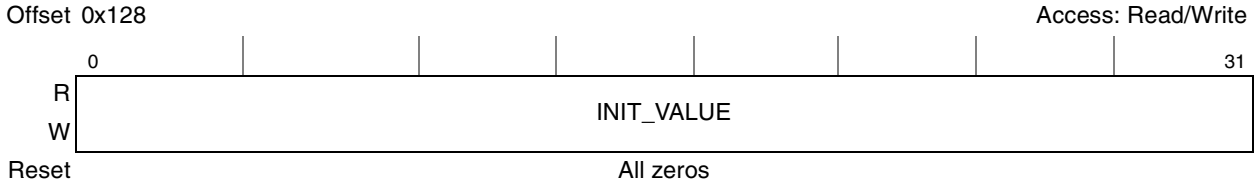


Figure 8-14. DDR SDRAM Data Initialization Configuration Register (DDR_DATA_INIT)

Table 8-19 describes the DDR_DATA_INIT fields.

Table 8-19. 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.

8.4.1.14 DDR SDRAM Clock Control (DDR_SDRAM_CLK_CNTL)

The DDR SDRAM clock control configuration register, shown in Figure 8-15, provides a 1/8-cycle clock adjustment.

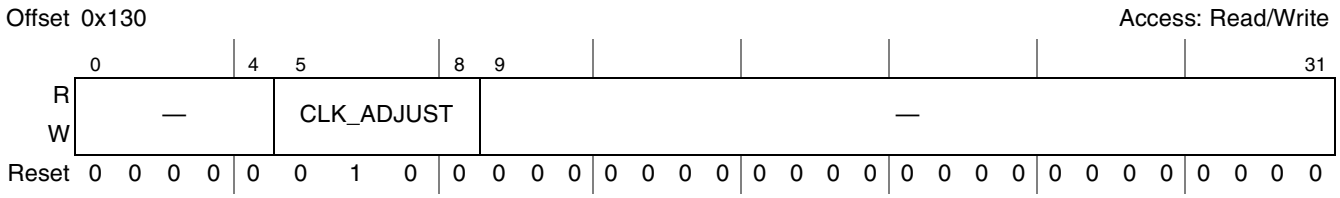


Figure 8-15. DDR SDRAM Clock Control Configuration Register (DDR_SDRAM_CLK_CNTL)

Table 8-20 describes the DDR_SDRAM_CLK_CNTL fields.

Table 8-20. DDR_SDRAM_CLK_CNTL Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–8	CLK_ADJUST	Clock adjust 0000 Clock is launched aligned with address/command 0001 Clock is launched 1/8 applied cycle after address/command 0010 Clock is launched 1/4 applied cycle after address/command 0011 Clock is launched 3/8 applied cycle after address/command 0100 Clock is launched 1/2 applied cycle after address/command 0101 Clock is launched 5/8 applied cycle after address/command 0110 Clock is launched 3/4 applied cycle after address/command 0111 Clock is launched 7/8 applied cycle after address/command 1000 Clock is launched 1 applied cycle after address/command 1001–1111 Reserved
9–31	—	Reserved

8.4.1.15 DDR Initialization Address (DDR_INIT_ADDR)

The DDR SDRAM initialization address register, shown in Figure 8-16, provides the address that is used for the data strobe to data skew adjustment and automatic $\overline{\text{CAS}}$ to preamble calibration after POR.

NOTE

After the skew adjustment, this address contains bad ECC data. This is not important at POR, as all of memory should be subsequently initialized if ECC is enabled (either by software or through the use of DDR_SDRAM_CFG_2[D_INIT]).

If an $\overline{\text{HRESET}}$ has been issued after the DRAM is in self-refresh mode, however, memory is not initialized, so this address should be written to using an 8- or 32-byte transaction to avoid possible ECC errors if this address could later be accessed.

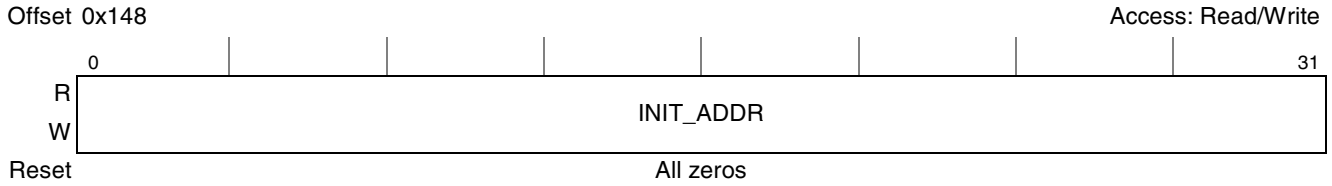


Figure 8-16. DDR Initialization Address Configuration Register (DDR_INIT_ADDR)

Table 8-21 describes the DDR_INIT_ADDR fields.

Table 8-21. DDR_INIT_ADDR Field Descriptions

Bits	Name	Description
0–31	INIT_ADDR	Initialization address. Represents the address that is used for the data strobe to data skew adjustment and automatic CAS to preamble calibration at POR. This address is written to during the initialization sequence.

8.4.1.16 DDR Initialization Enable Extended Address (DDR_INIT_EXT_ADDR)

The DDR SDRAM initialization extended address register, shown in Figure 8-17, provides the extended address that is used for the data strobe to data skew adjustment and automatic $\overline{\text{CAS}}$ to preamble calibration after POR.

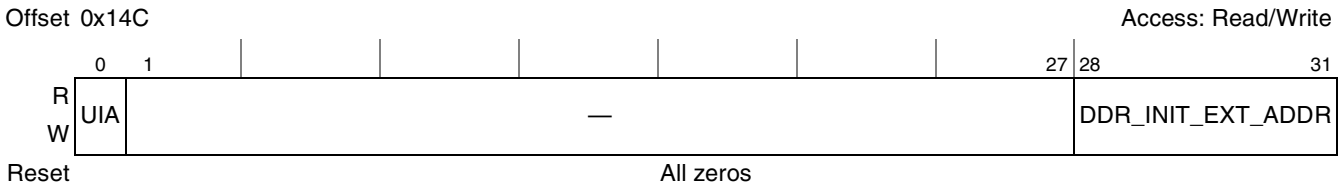


Figure 8-17. DDR Initialization Extended Address Configuration Register (DDR_INIT_EXT_ADDR)

Table 8-22 describes the DDR_INIT_EXT_ADDR fields.

Table 8-22. DDR_INIT_EXT_ADDR Field Descriptions

Bits	Name	Description
0	UIA	Use initialization address 0 Use the default address for training sequence as calculated by the controller. This is the first valid address in the first enabled chip select. 1 Use the initialization address programmed in DDR_INIT_ADDR and DDR_INIT_EXT_ADDR.
1–27	—	Reserved
28–31	INIT_EXT_ADDR	Initialization extended address Represents the extended address that is used for the data strobe to data skew adjustment and automatic $\overline{\text{CAS}}$ to preamble calibration at POR. This extended address is written to during the initialization sequence.

8.4.1.17 DDR Debug Status Register 1 (DDRDSR_1)

The DDRDSR_1 register, shown in Figure 8-18, contains the DDR driver compensation input value and the current settings of the P and N FET impedance for MDIC*n*, command/control, and data.

Offset 0xB20 Access: Read only

	0 1	2 5	6 9	10 15	16 19	20 23	24 27	28 31
R	DDRDC	MDICPZ	MDICNZ	—	CPZ	CNZ	DPZ	DNZ
W								
Reset	All zeros							

Figure 8-18. DDR Debug Status Register 1 (DDRDSR_1)

Table 8-23 describes the DDRDSR_1 fields.

Table 8-23. DDRDSR_1 Field Descriptions

Bits	Name	Description
0–1	DDRDC	DDR driver compensation input value
2–5	MDICPZ	Current setting of PFET driver MDIC impedance
6–9	MDICNZ	Current setting of NFET driver MDIC impedance
10–15	—	Reserved, should be cleared.
16–19	CPZ	Current setting of PFET driver command impedance
20–23	CNZ	Current setting of NFET driver command impedance
24–27	DPZ	Current setting of PFET driver data impedance
28–31	DNZ	Current setting of NFET driver data impedance

8.4.1.18 DDR Debug Status Register 2 (DDRDSR_2)

The DDRDSR_2 register, shown in Figure 8-19, contains the current settings of the P and N FET impedance for the DDR drivers for clocks.

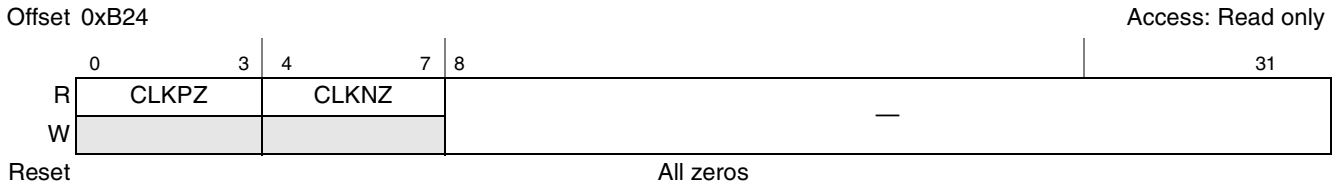


Figure 8-19. DDR Debug Status Register 2 (DDRDSR_2)

Table 8-24 describes the DDRDSR_2 fields.

Table 8-24. DDRDSR_2 Field Descriptions

Bits	Name	Description
0–3	CLKPZ	Current setting of PFET driver clock impedance
4–7	CLKNZ	Current setting of NFET driver clock impedance
8–31	—	Reserved

8.4.1.19 DDR Control Driver Register 1 (DDRCDR_1)

DDRCDR_1, shown in Figure 8-20, sets the driver hardware compensation enable, the DDR MDIC driver P/N impedance, ODT termination value for IOs, driver software override enable for MDIC, driver software override enable for address/command, driver software override enable for data, the DDR address/command driver P/N impedance, and the DDR data driver P/N impedance.

The fields in DDRCDR_1, other than DDRCDR_1[ODT], are used to enable driver calibration with the MDIC[0:1] pins. This can be used to calibrate the DDR drivers to 18 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 by setting DDRCDR_1[DHC_EN].

NOTE

All driver calibration, whether by software or hardware, should be done before the DDR controller is enabled (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_1[DSO_MDIC_EN] and ensure that DDRCDR_1[DHC_EN] is cleared
2. Set the highest impedance (value 0000) for DDRCDR_1[DSO_MDICPZ]
3. Set DDRCDR_1[DSO_MDIC_PZ_OE] to enable the output enable for MDIC[0]
4. After at least 4 cycles, read DDRDSR_1[0]. If the value is 0, then use the next lowest impedance, and read DDRDSR_1[0] again. Once a value of 1 is detected, then leave DDRCDR_1[DSO_MDICPZ] at the calibrated value

5. Clear DDRCDR_1[DSO_MDIC_PZ_OE]
6. After DDRCDR_1[DSO_MDICPZ} is calibrated, set a value of 0000 for DDRCDR_1[DSO_MDICNZ]
7. Set DDRCDR_1[DSO_MDIC_NZ_OE] to enable the output enable for MDIC[1]
8. After at least 4 cycles, read DDRDSR_1[1]. If the value is 1, then use the next lowest impedance, and read DDRDSR_1[1] again. Once a value of 0 is detected, then leave DDRCDR_1[DSO_MDICNZ] at the calibrated value
9. Clear DDRCDR_1[DSO_MDIC_NZ_OE]

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

- 0000
- 1000
- 1100
- 1110
- 1111

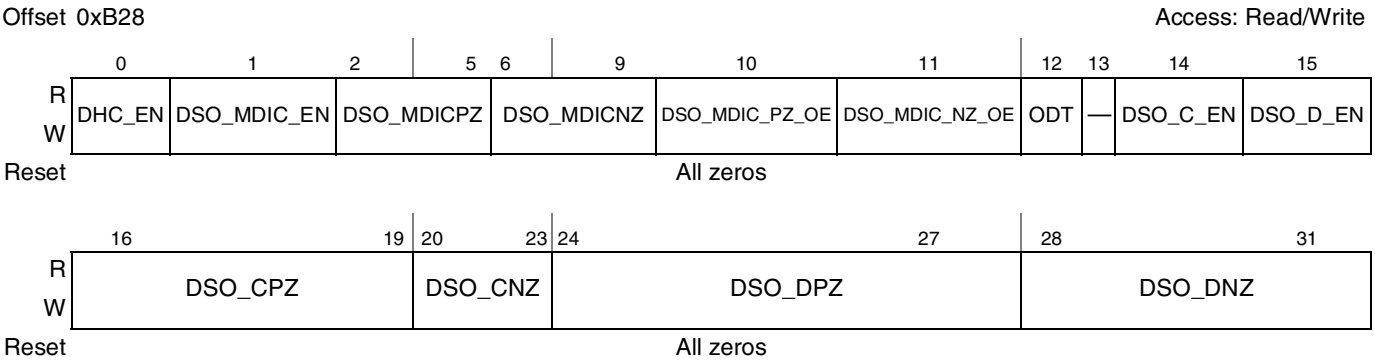


Figure 8-20. DDR Control Driver Register 1 (DDRCDR_1)

Table 8-25 describes the DDRCDR_1 fields.

Table 8-25. DDRCDR_1 Field Descriptions

Bits	Name	Description
0	DHC_EN	DDR driver hardware compensation enable
1	DSO_MDIC_EN	Driver software override enable for MDIC
2–5	DSO_MDICPZ	DDR driver software MDIC p-impedance override
6–9	DSO_MDICNZ	DDR driver software MDIC n-impedance override
10	DSO_MDIC_PZ_OE	Driver software override p-impedance output enable
11	DSO_MDIC_NZ_OE	Driver software override n-impedance output enable
12	ODT	ODT termination value for IOs 0 75 Ω 1 150 Ω
13	—	Reserved

Table 8-25. DDRCDR_1 Field Descriptions (continued)

Bits	Name	Description
14	DSO_C_EN	Driver software override enable for address/command
15	DSO_D_EN	Driver software override enable for data
16–19	DSO_CPZ	DDR driver software command p-impedance override
20–23	DSO_CNZ	DDR driver software command n-impedance override
24–27	DSO_DPZ	Driver software data p-impedance override
28–31	DSO_DNZ	Driver software data n-impedance override

8.4.1.20 DDR Control Driver Register 2 (DDRCDR_2)

The DDRCDR_2, shown in Figure 8-21, sets the driver software override enable for clocks and the DDR clocks driver P/N impedance.

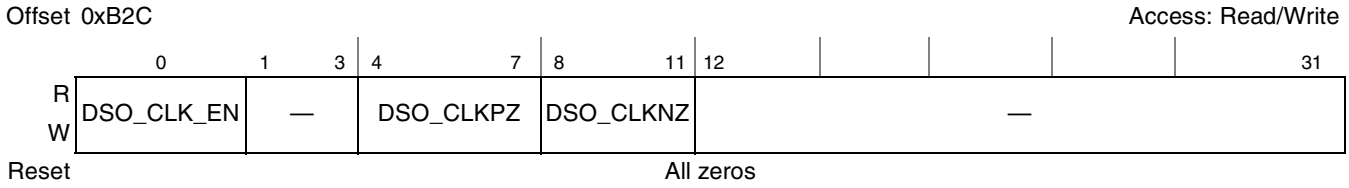


Figure 8-21. DDR Control Driver Register 2 (DDRCDR_2)

Table 8-26 describes the DDRCDR_2 fields.

Table 8-26. DDRCDR_2 Field Descriptions

Bits	Name	Description
0	DSO_CLK_EN	Driver software override enable for clocks
1–3	—	Reserved
4–7	DSO_CLKPZ	Driver software clocks p-impedance override
8–11	DSO_CLKNZ	Driver software clocks n-impedance override
12–31	—	Reserved

8.4.1.21 DDR IP Block Revision 1 (DDR_IP_REV1)

The DDR IP block revision 1 register, shown in Figure 8-22, provides read-only fields with the IP block ID, along with major and minor revision information.

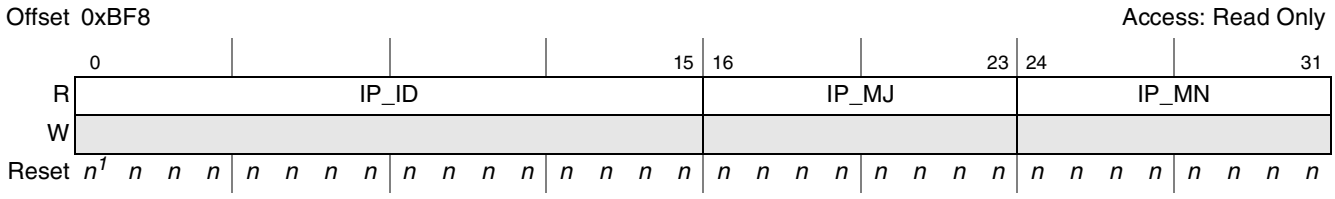


Figure 8-22. DDR IP Block Revision 1 (DDR_IP_REV1)

[1] For reset values, see Table 8-27.

Table 8-27 describes the DDR_IP_REV1 fields.

Table 8-27. 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 0x03.
24–31	IP_MN	Minor revision. This is currently set to 0x02.

8.4.1.22 DDR IP Block Revision 2 (DDR_IP_REV2)

The DDR IP block revision 2 register, shown in Figure 8-23, provides read-only fields with the IP block integration and configuration options.

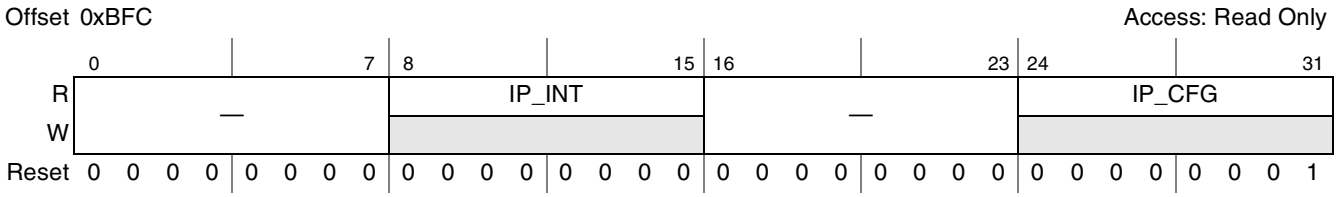


Figure 8-23. DDR IP Block Revision 2 (DDR_IP_REV2)

Table 8-28 describes the DDR_IP_REV2 fields.

Table 8-28. 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

8.4.1.23 Memory Data Path Error Injection Mask High (DATA_ERR_INJECT_HI)

The memory data path error injection mask high register is shown in Figure 8-24.

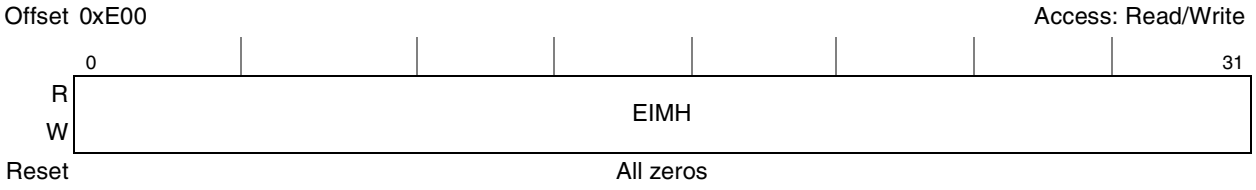


Figure 8-24. Memory Data Path Error Injection Mask High Register (DATA_ERR_INJECT_HI)

Table 8-29 describes the DATA_ERR_INJECT_HI fields.

Table 8-29. 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.

8.4.1.24 Memory Data Path Error Injection Mask Low (DATA_ERR_INJECT_LO)

The memory data path error injection mask low register is shown in Figure 8-25.

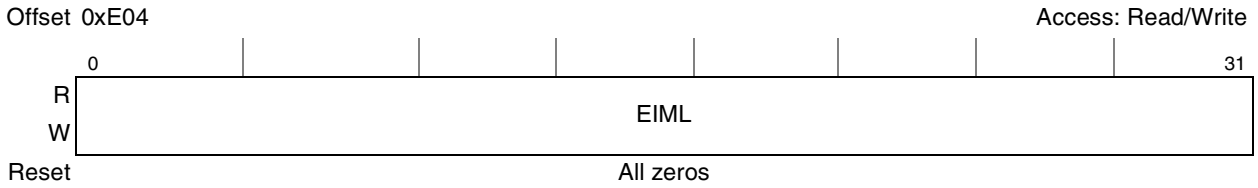


Figure 8-25. Memory Data Path Error Injection Mask Low Register (DATA_ERR_INJECT_LO)

Table 8-30 describes the DATA_ERR_INJECT_LO fields.

Table 8-30. 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.

8.4.1.25 Memory Data Path Error Injection Mask ECC (ERR_INJECT)

The memory data path error injection mask ECC register, shown in Figure 8-26, sets the ECC mask, enables errors to be written to ECC memory, and allows the ECC byte to mirror the most significant data byte.

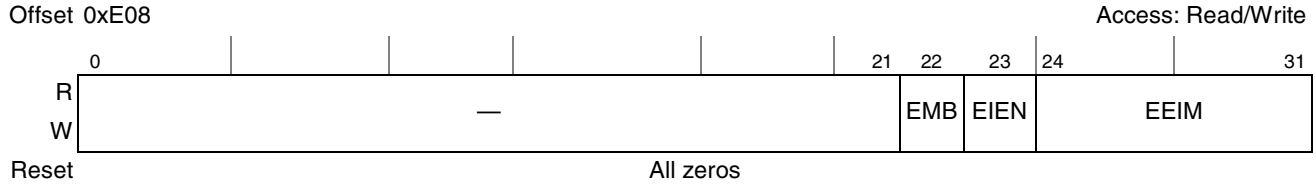


Figure 8-26. Memory Data Path Error Injection Mask ECC Register (ERR_INJECT)

Table 8-31 describes the ERR_INJECT fields.

Table 8-31. 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.

8.4.1.26 Memory Data Path Read Capture High (CAPTURE_DATA_HI)

The memory data path read capture high register, shown in Figure 8-27, stores the high word of the read data path during error capture.

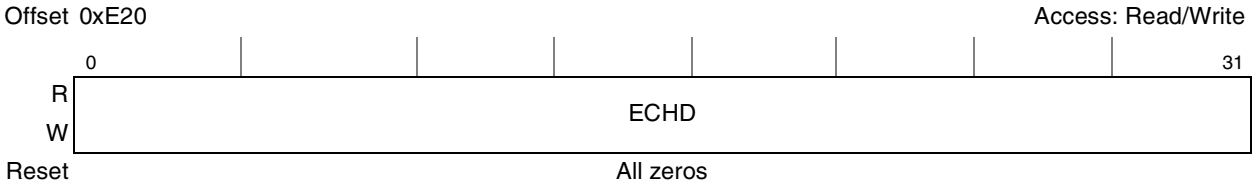


Figure 8-27. Memory Data Path Read Capture High Register (CAPTURE_DATA_HI)

Table 8-32 describes the CAPTURE_DATA_HI fields.

Table 8-32. 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.

8.4.1.27 Memory Data Path Read Capture Low (CAPTURE_DATA_LO)

The memory data path read capture low register, shown in Figure 8-28, stores the low word of the read data path during error capture.

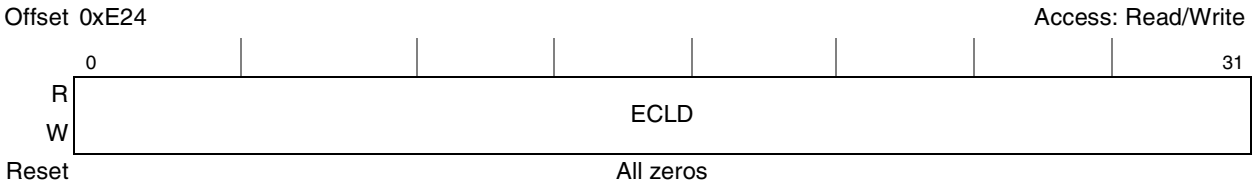


Figure 8-28. Memory Data Path Read Capture Low Register (CAPTURE_DATA_LO)

Table 8-33 describes the CAPTURE_DATA_LO fields.

Table 8-33. 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.

8.4.1.28 Memory Data Path Read Capture ECC (CAPTURE_ECC)

The memory data path read capture ECC register, shown in Figure 8-29, stores the ECC syndrome bits that were on the data bus when an error was detected.

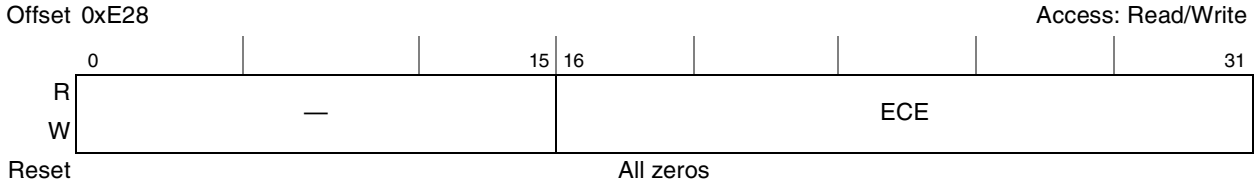


Figure 8-29. Memory Data Path Read Capture ECC Register (CAPTURE_ECC)

Table 8-34 describes the CAPTURE_ECC fields.

Table 8-34. 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.

8.4.1.29 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 8-30.

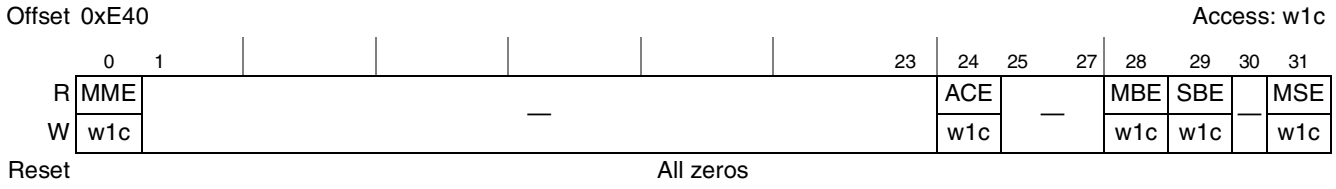


Figure 8-30. Memory Error Detect Register (ERR_DETECT)

Table 8-35 describes the ERR_DETECT fields.

Table 8-35. 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.

8.4.1.30 Memory Error Disable (ERR_DISABLE)

The memory error disable register, shown in Figure 8-31, allows selective disabling of the DDR controller's error detection circuitry. Disabled errors are not detected or reported.

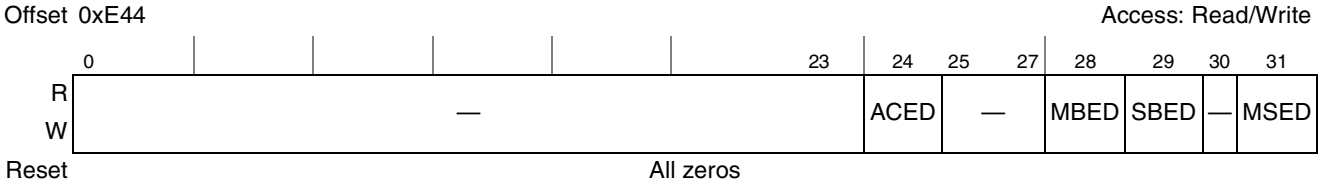


Figure 8-31. Memory Error Disable Register (ERR_DISABLE)

Table 8-36 describes the ERR_DISABLE fields.

Table 8-36. 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

Table 8-36. ERR_DISABLE Field Descriptions (continued)

Bits	Name	Description
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. 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.

8.4.1.31 Memory Error Interrupt Enable (ERR_INT_EN)

The memory error interrupt enable register, shown in Figure 8-32, enables ECC interrupts or memory select error interrupts. When an enabled interrupt condition occurs, the internal *int* signal is asserted to the programmable interrupt controller (PIC).

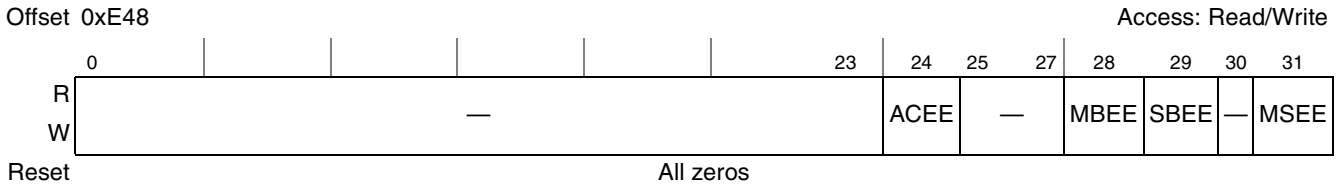


Figure 8-32. Memory Error Interrupt Enable Register (ERR_INT_EN)

Table 8-37 describes the ERR_INT_EN fields.

Table 8-37. 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. Note that non-correctable read errors may cause an interrupt. 0 Multiple-bit ECC errors cannot generate interrupts. 1 Multiple-bit ECC errors generate interrupts.
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.

8.4.1.32 Memory Error Attributes Capture (CAPTURE_ATTRIBUTES)

The memory error attributes capture register, shown in Figure 8-33, sets attributes for errors including type, size, source, and others.

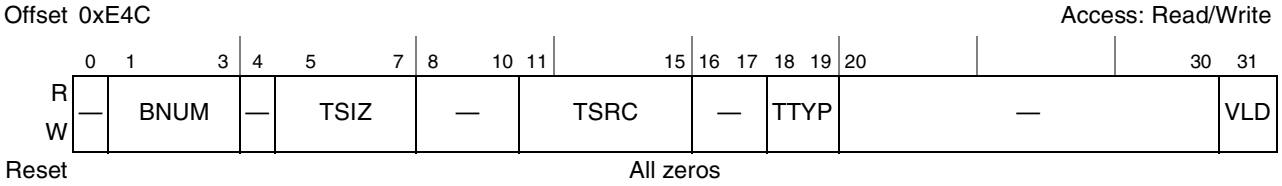


Figure 8-33. Memory Error Attributes Capture Register (CAPTURE_ATTRIBUTES)

Table 8-38 describes the CAPTURE_ATTRIBUTES fields.

Table 8-38. CAPTURE_ATTRIBUTES Field Descriptions

Bits	Name	Description
0	—	Reserved
1–3	BNUM	Data beat number. Captures the double-word 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
11–15	TSRC	Transaction source for the error 00000 PCI 00001 Reserved 00010 PCI Express 1 00011 PCI Express 2 00100 Local bus controller 00101 Reserved 00110 Reserved 00111 Reserved 01000 Configuration space 01001 LCD controller 01010 Boot sequencer 01011 Reserved 01100 Reserved 01101 Reserved 01110 Reserved 01111 DDR memory controller 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 Reserved 10101 DMA1 10110 DMA2 10111 SAP (System access port) 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 Reserved
16–17	—	Reserved

Table 8-38. CAPTURE_ATTRIBUTES Field Descriptions (continued)

Bits	Name	Description
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.

8.4.1.33 Memory Error Address Capture (CAPTURE_ADDRESS)

The memory error address capture register, shown in Figure 8-34, holds the 32 lsbs of a transaction when a DDR ECC error is detected.

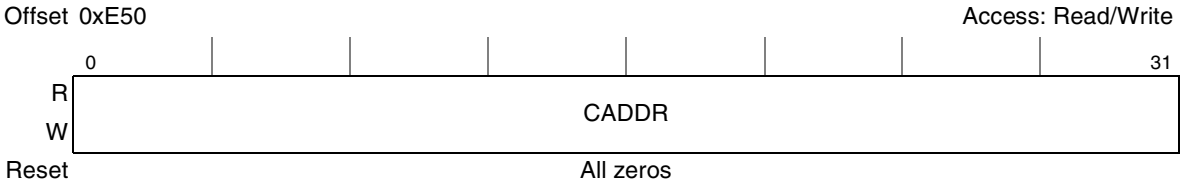


Figure 8-34. Memory Error Address Capture Register (CAPTURE_ADDRESS)

Table 8-39 describes the CAPTURE_ADDRESS fields.

Table 8-39. 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.

8.4.1.34 Memory Error Extended Address Capture (CAPTURE_EXT_ADDRESS)

The memory error extended address capture register, shown in Figure 8-35, holds the four most significant transaction bits when an error is detected.

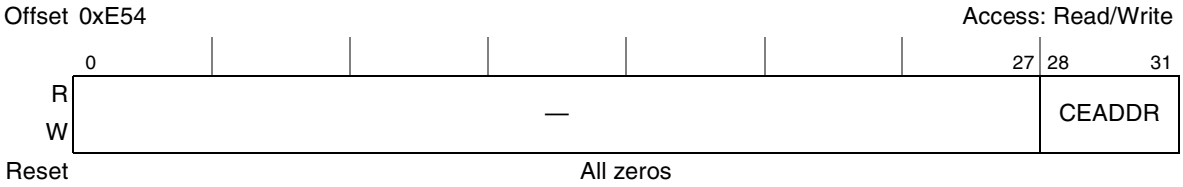


Figure 8-35. Memory Error Extended Address Capture Register (CAPTURE_EXT_ADDRESS)

Table 8-40 describes the CAPTURE_EXT_ADDRESS fields.

Table 8-40. CAPTURE_EXT_ADDRESS Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28–31	CEADDR	Captured extended address. Captures the 4 msbs of the transaction address when an error is detected

8.4.1.35 Single-Bit ECC Memory Error Management (ERR_SBE)

The single-bit ECC memory error management register, shown in Figure 8-36, 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.

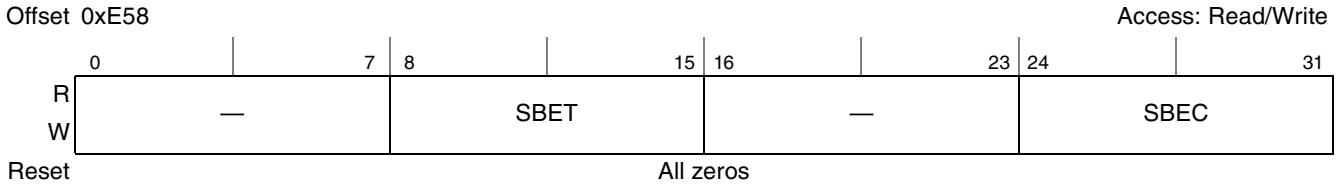


Figure 8-36. Single-Bit ECC Memory Error Management Register (ERR_SBE)

Table 8-41 describes the ERR_SBE fields.

Table 8-41. 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.

8.5 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 8-37 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 32-/40-bit wide memory. Bank sizes up to 4 Gbits are supported, providing up to a maximum of 16 Gbits of DDR main memory per chip select.

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].

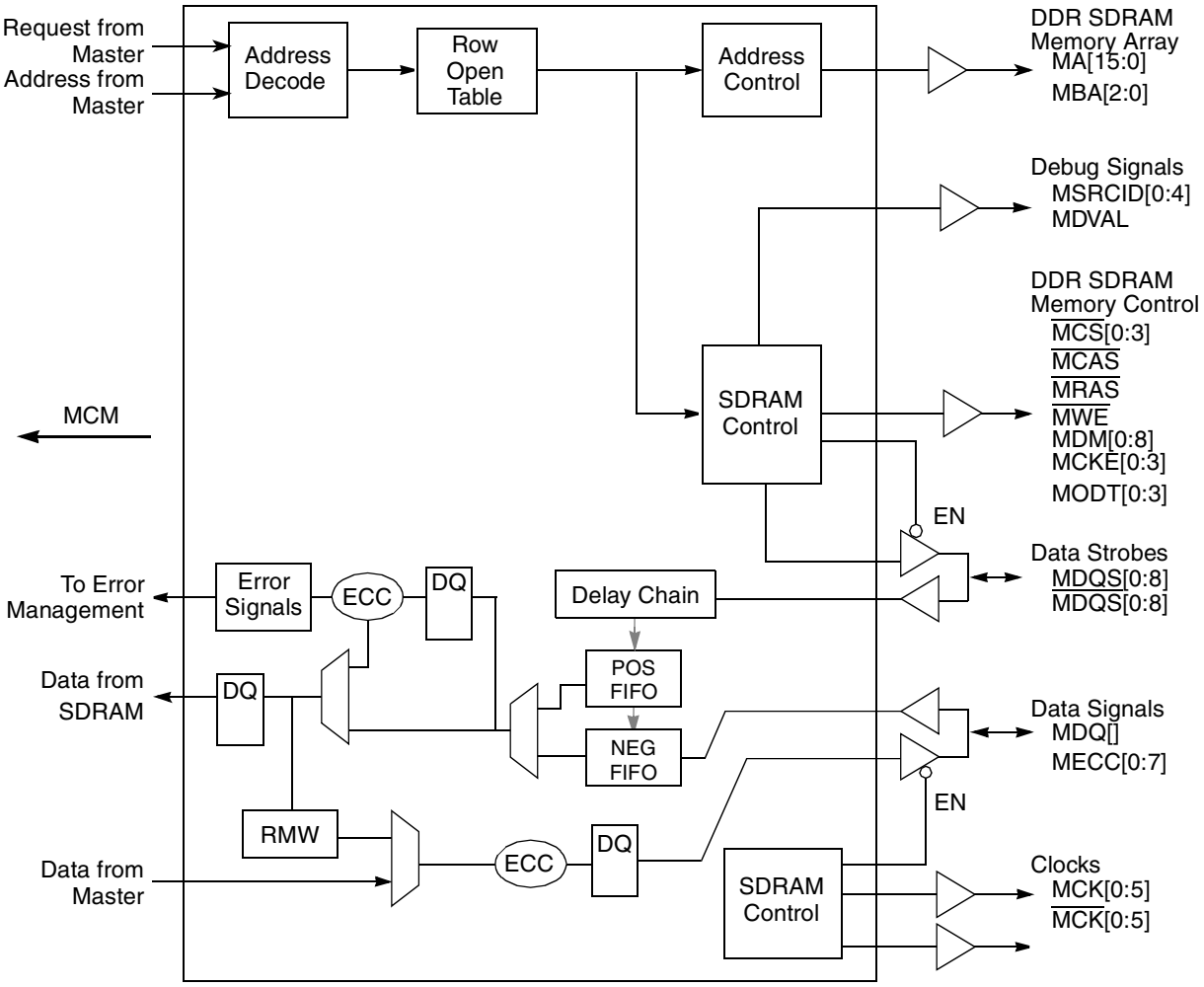


Figure 8-37. DDR Memory Controller Block Diagram

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[]) 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 8-38 shows an example DDR SDRAM configuration with four logical banks.

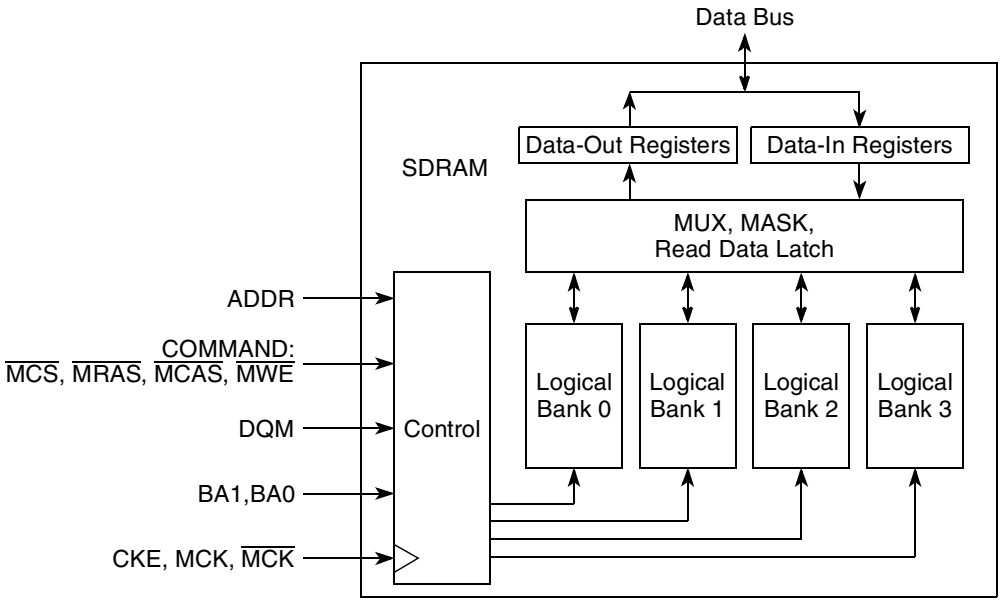


Figure 8-38. Typical Dual Data Rate SDRAM Internal Organization

Figure 8-39 shows some typical signal connections.

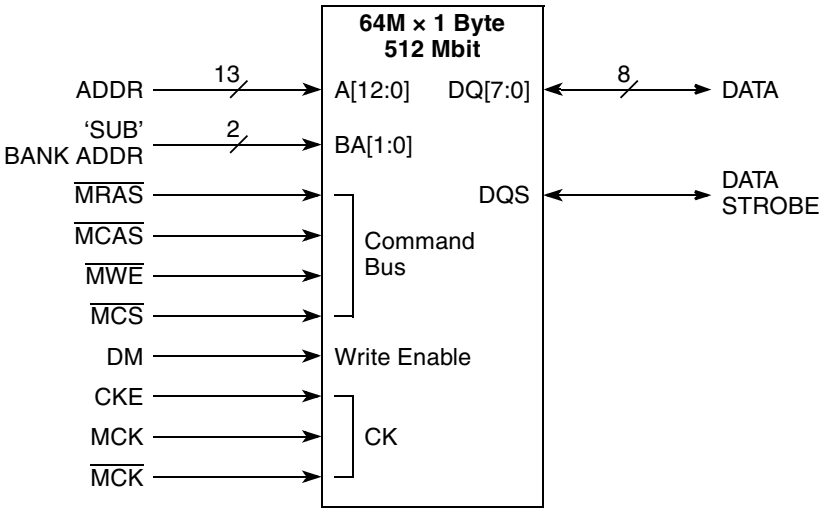
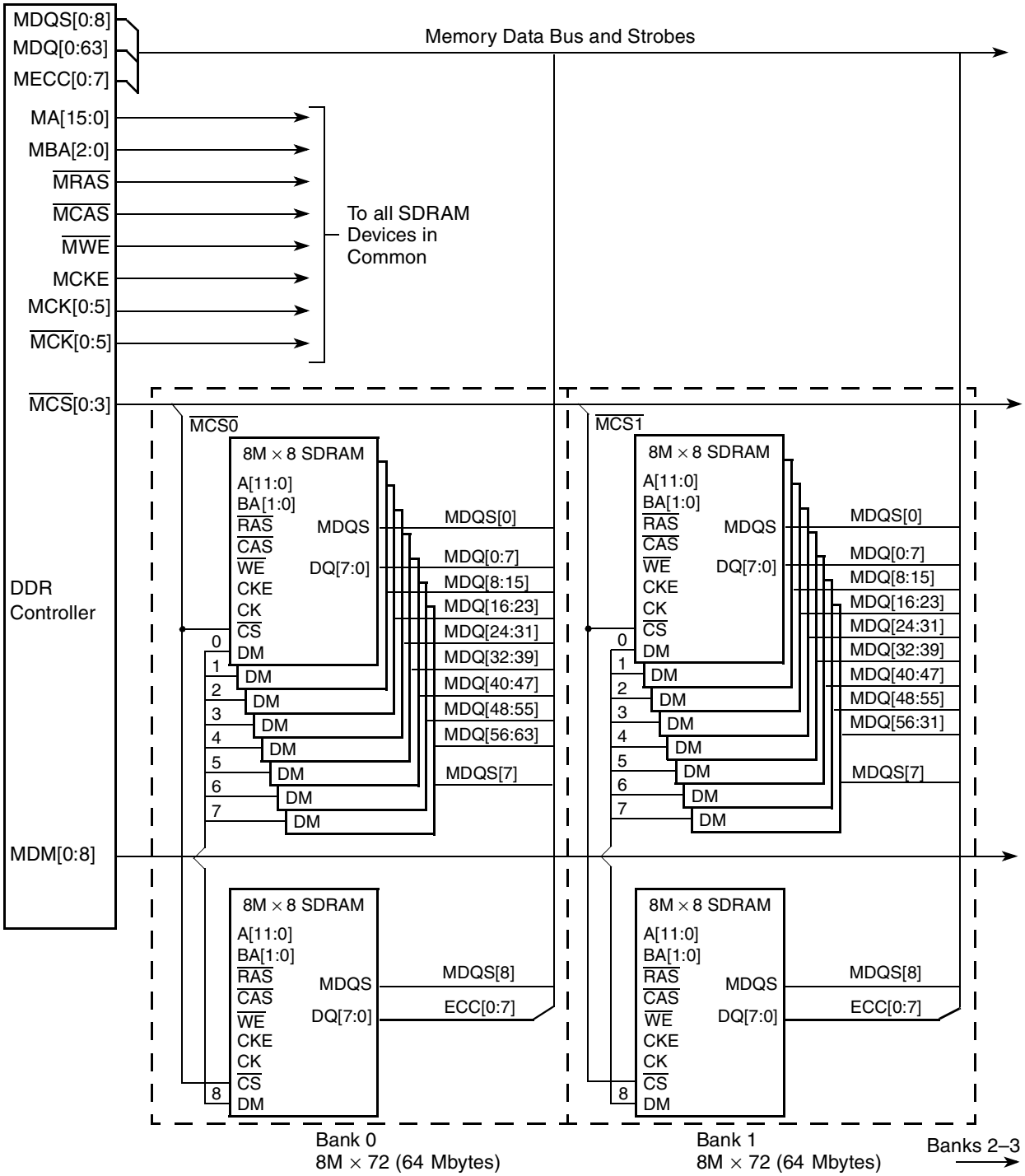


Figure 8-39. Typical DDR SDRAM Interface Signals

Figure 8-40 shows an example DDR SDRAM configuration with four physical banks each comprised of 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 16 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 $\overline{\text{MCS}}$[0:3], MCK*n*, MDM[0:8], and the data bus signals.
2. Each of the $\overline{\text{MCS}}$[0:3] signals correspond with a separate physical bank of memory.
3. Buffering may be needed if large memory arrays are used.
4. MCK*n* may be apportioned among all memory devices. Complementary bus is not shown.

Figure 8-40. Example 256-Mbyte DDR SDRAM Configuration With ECC

Section 8.5.12, "Error Management," explains how the DDR memory controller handles errors.

8.5.1 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. Sixteen 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 8-42 shows the DDR memory controller's relationships between data byte lane, MDM[], MDQS[], and MDQ[] when DDR SDRAM memories are used with ×8 or ×16 devices.

Table 8-42. Byte Lane to Data Relationship

Data Byte Lane	Data Bus Mask	Data Bus Strobe	Data Bus
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]

8.5.1.1 Supported DDR SDRAM Organizations

Although the DDR memory controller multiplexes row and column address bits onto 16 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 16 row address bits, plus 2 or 3 logical bank-select bits and from 8–11 column address bits.

Table 8-43, Table 8-44, and Table 8-45 describe DDR SDRAM device configurations supported by the DDR memory controller.

NOTE

DDR SDRAM is limited to 30 total address bits.

Table 8-43. Supported DDR1 SDRAM Device Configurations

SDRAM Device	Device Configuration	Row x Column x Sub-Bank Bits	-Bit Bank Size	Four Banks of Memory
64 Mbits	8 Mbits x 8	12 x 9 x 2	Mbytes	Mbytes
64 Mbits[1]	4 Mbits x 16	12 x 8 x 2	Mbytes	Mbytes
128 Mbits	16 Mbits x 8	12 x 10 x 2	Mbytes	Mbytes
128 Mbits	8 Mbits x 16	12 x 9 x 2	Mbytes	Mbytes
256 Mbits	32 Mbits x 8	13 x 10 x 2	Mbytes	
256 Mbits	16 Mbits x 16	13 x 9 x 2	Mbytes	Mbytes
512 Mbits	64 Mbits x 8	13 x 11 x 2	Mbytes	Gbyte
512 Mbits	32 Mbits x 16	13 x 10 x 2	Mbytes	
1 Gbit	128 Mbits x 8	14 x 11 x 2		Gbytes
1 Gbit	64 Mbits x 16	14 x 10 x 2	Mbytes	Gbyte
2 Gbits	256 Mbits x 8	15 x 11 x 2	Gbyte	Gbytes
2 Gbits	128 Mbits x 16	15 x 10 x 2		Gbytes
4 Gbits	512 Mbits x 8	16 x 11 x 2	Gbytes	Gbytes
4 Gbits	256 Mbits x 16	16 x 10 x 2	Gbyte	Gbytes

[1] This configuration is not supported in 16-bit bus mode.

Table 8-44. Supported DDR2 SDRAM Device Configurations

SDRAM Device	Device Configuration	Row x Column x Sub-Bank Bits	-Bit Bank Size	Four Banks of Memory
256 Mbits	32 Mbits x 8	13 x 10 x 2	Mbytes	
256 Mbits	16 Mbits x 16	13 x 9 x 2	Mbytes	Mbytes
512 Mbits	64 Mbits x 8	14 x 10 x 2	Mbytes	Gbyte
512 Mbits	32 Mbits x 16	13 x 10 x 2	Mbytes	
1 Gbit	128 Mbits x 8	14 x 10 x 3		Gbytes
1 Gbit	64 Mbits x 16	13 x 10 x 3	Mbytes	Gbyte
2 Gbits	256 Mbits x 8	15 x 10 x 3	Gbyte	Gbytes
2 Gbits	128Mbits x 16	14 x 10 x 3		Gbytes
4 Gbits	512 Mbits x 8	16 x 10 x 3	Gbytes	Gbytes
4 Gbits	256 Mbits x 16	15 x 10 x 3	Gbyte	Gbytes

Table 8-45. Supported DDR2 SDRAM Device Configurations—One Physical Bank

SDRAM Device	Device Configuration	Row × Column × Sub-bank Bits	16-Bit Bank Size
256 Mbits	32 Mbits × 8	13 × 10 × 2	64 Mbytes
256 Mbits	16 Mbits × 16	13 × 9 × 2	32 Mbytes
512 Mbits	64 Mbits × 8	14 × 10 × 2	128 Mbytes
512 Mbits	32 Mbits × 16	13 × 10 × 2	64 Mbytes
1 Gbits	128 Mbits × 8	14 × 10 × 3	256 Mbytes
1 Gbits	64 Mbits × 16	13 × 10 × 3	128 Mbytes
2 Gbits	256Mbits × 8	14 × 11 × 3	512 Mbytes
2 Gbits	128Mbits × 16	14 × 10 × 3	256 Mbytes

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 8.5.12, “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 $\overline{\text{MCS}}n$ 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.

8.5.2 DDR SDRAM Address Multiplexing

Table 8-46, Table 8-47, Table 8-48Table 8-49, Table 8-50 show the address bit encodings for each DDR SDRAM configuration. The address presented at the memory controller signals MA[15:0] use MA[15] 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 8-46. DDR1 Address Multiplexing for 64-Bit Data Bus with Interleaving Disabled

Row x Col		msb	Address from Core Master																											lsb
		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	32	33–35
16 x 11 x 2	$\overline{\text{MRAS}}$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{\text{MCAS}}$																			11	9	8	7	6	5	4	3	2	1	0	
16 x 10 x 2	$\overline{\text{MRAS}}$		15	14	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	
15 x 11 x 2	$\overline{\text{MRAS}}$		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{\text{MCAS}}$																			11	9	8	7	6	5	4	3	2	1	0	

Table 8-46. DDR1 Address Multiplexing for 64-Bit Data Bus with Interleaving Disabled (continued)

Row x Col		msb	Address from Core Master																												lsb
		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	32	33–35
15 x 10 x 2	$\overline{\text{MRAS}}$			14	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	
14 x 11 x 2	$\overline{\text{MRAS}}$			13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{\text{MCAS}}$																			11	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 11 x 2	$\overline{\text{MRAS}}$				12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																	1	0												
	$\overline{\text{MCAS}}$																			11	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	
12 x 10 x 2	$\overline{\text{MRAS}}$						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	
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 8-47. DDR1 Address Multiplexing for 32-Bit Data Bus with Interleaving Disabled

Row x Col		msb	Address from Core Master																													lsb
		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	32	33	34–35
16 x 11 x 2	$\overline{\text{MRAS}}$		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	$\overline{\text{MCAS}}$																				11	9	8	7	6	5	4	3	2	1	0	

Table 8-47. DDR1 Address Multiplexing for 32-Bit Data Bus with Interleaving Disabled (continued)

Row x Col		msb	Address from Core Master																												lsb
		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	32	33	34–35
16 x 10 x 2	$\overline{\text{MRAS}}$			15	14	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	
15 x 11 x 2	$\overline{\text{MRAS}}$			14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	$\overline{\text{MCAS}}$																				11	9	8	7	6	5	4	3	2	1	0	
15 x 10 x 2	$\overline{\text{MRAS}}$				14	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	
14 x 11 x 2	$\overline{\text{MRAS}}$				13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	$\overline{\text{MCAS}}$																				11	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 11 x 2	$\overline{\text{MRAS}}$					12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																		1	0												
	$\overline{\text{MCAS}}$																				11	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	
12 x 10 x 2	$\overline{\text{MRAS}}$							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	
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 8-48. DDR1 Address Multiplexing for 16-Bit Data Bus

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	$\overline{\text{MRAS}}$				14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
	MBA																			1	0											
	$\overline{\text{MCAS}}$																					11	9	8	7	6	5	4	3	2	1	0
15 x 10 x 2	$\overline{\text{MRAS}}$					14	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
14 x 11 x 2	$\overline{\text{MRAS}}$					13	12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{\text{MCAS}}$																					11	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 11 x 2	$\overline{\text{MRAS}}$						12	11	10	9	8	7	6	5	4	3	2	1	0													
	MBA																			1	0											
	$\overline{\text{MCAS}}$																					11	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
12 x 10 x 2	$\overline{\text{MRAS}}$								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
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 8-49. 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
		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	32	33	34–35
16 x 10 x 3	$\overline{MRAS}$		15	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	
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	

Table 8-50. DDR2 Address Multiplexing for 16-Bit Data Bus

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	

Table 8-50. DDR2 Address Multiplexing for 16-Bit Data Bus (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
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	

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 8-51 illustrates examples of address decode when interleaving between two chip selects, and Table 8-52 shows examples of address decode when interleaving between four chip selects.

Table 8-51. Example of Address Multiplexing for -Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled

Row x Col		msb	Address from Core Master																														lsb
		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	32	33–35
14 x 10 x 3	$\overline{\text{MRAS}}$		13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																
	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	CS SEL															
	MBA																		1	0												
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0		

Table 8-51. Example of Address Multiplexing for -Bit Data Bus Interleaving between Two Banks with Partial Array Self Refresh Disabled (continued)

Row x Col		msb	Address from Core Master																												lsb
		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	32	33–35
13 x 10 x 3	$\overline{\text{MRAS}}$			12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL														
	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	CS SEL													
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	

8.5.3 JEDEC Standard DDR SDRAM Interface Commands

Table 8-52. 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
		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	32	33–35
14 x 10 x 3	$\overline{\text{MRAS}}$	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CS SEL																
	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	CS SEL														
	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	CS SEL																
	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	CS SEL														
	MBA																		1	0											
	$\overline{\text{MCAS}}$																				9	8	7	6	5	4	3	2	1	0	

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 8-53) 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 DDR1 memory 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 8-53. 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

8.5.4 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 MDM[]. 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 8-54 with granularity of one memory clock cycle, except for CASLAT, which can be programmed with ½ clock granularity.

Table 8-54. 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}.

Table 8-54. DDR SDRAM Interface Timing Intervals (continued)

Timing Intervals	Definition
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 8.4.1.4, “DDR SDRAM Timing Configuration 0 (TIMING_CFG_0),” Section 8.4.1.5, “DDR SDRAM Timing Configuration 1 (TIMING_CFG_1),” Section 8.4.1.6, “DDR SDRAM Timing Configuration 2 (TIMING_CFG_2),” and Section 8.4.1.3, “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 8-41 through Figure 8-43 show DDR SDRAM timing for various types of accesses; see Figure 8-41 for a single-beat read operation, Figure 8-42 for a single-beat write operation, and Figure 8-43 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).

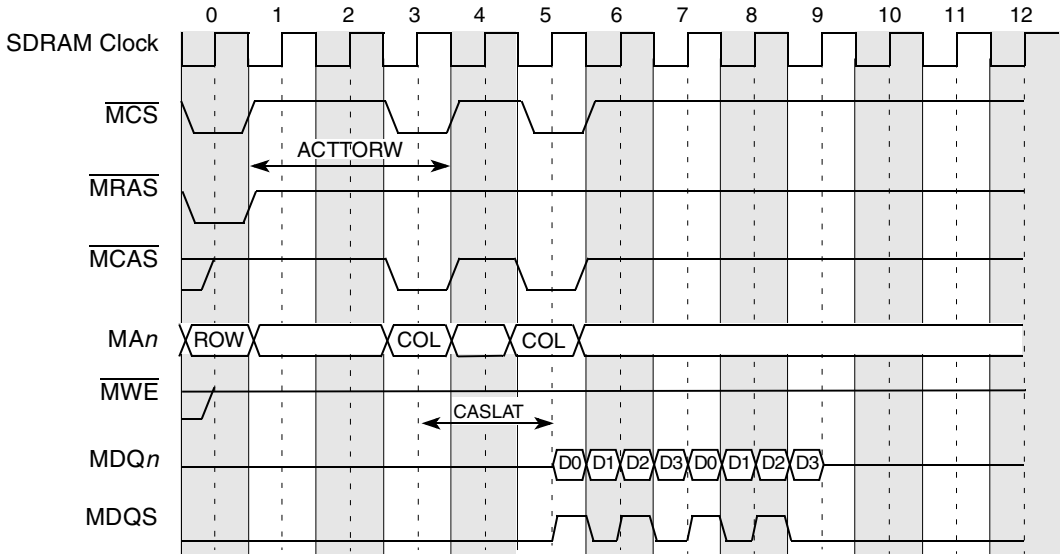


Figure 8-41. DDR SDRAM Burst Read Timing—ACTTORW = 3, MCAS Latency = 2

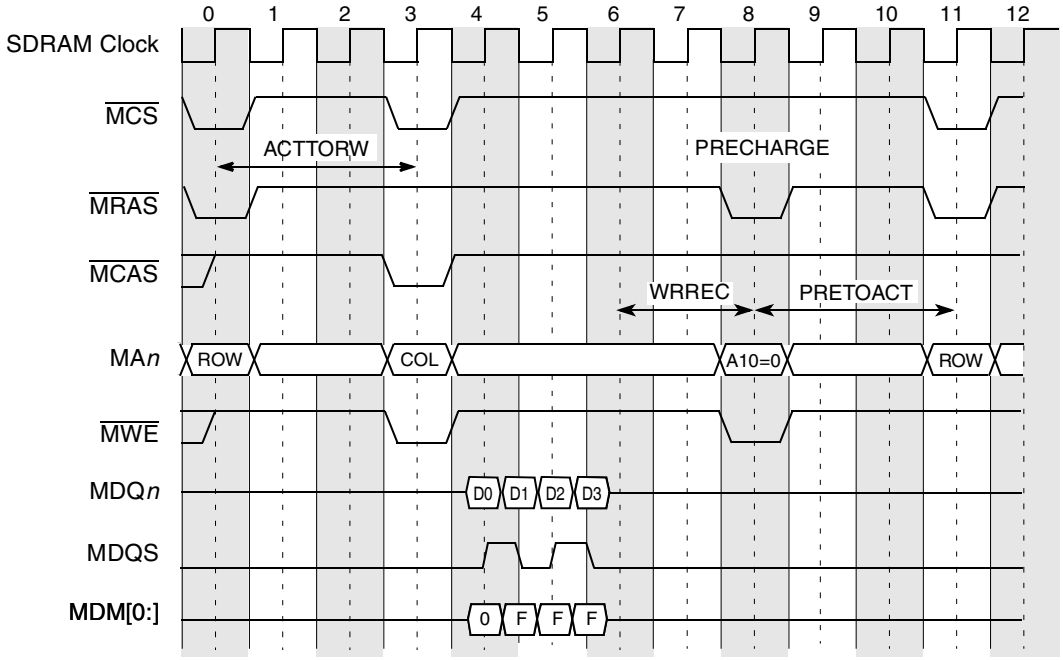


Figure 8-42. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTOR

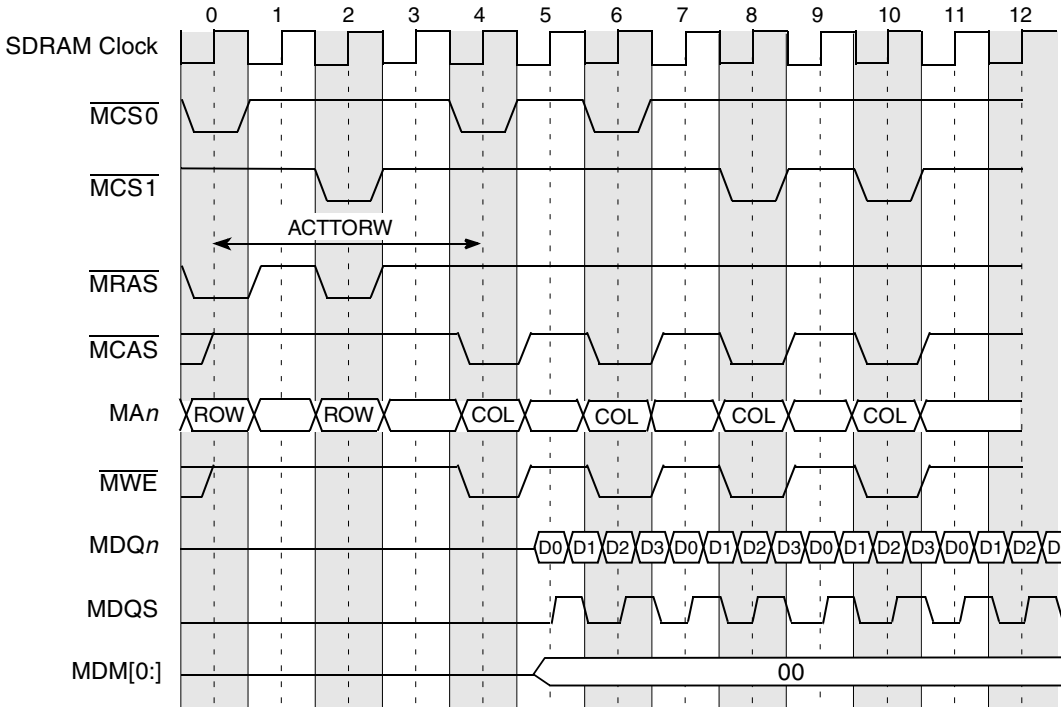


Figure 8-43. DDR SDRAM Single-Beat (Double Word) Write Timing—ACTTORW = 3

8.5.4.1 Clock Distribution

Clock distribution has the following recommendations:

- 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.

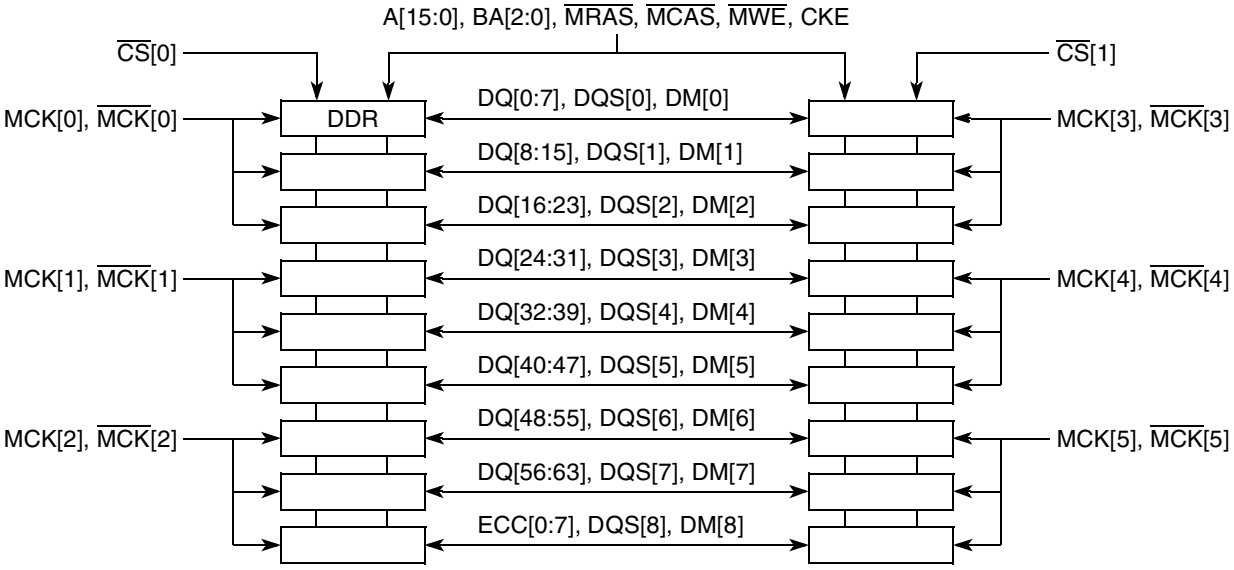


Figure 8-44. DDR SDRAM Clock Distribution Example for x8 DDR SDRAMs

8.5.5 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 8-45 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.

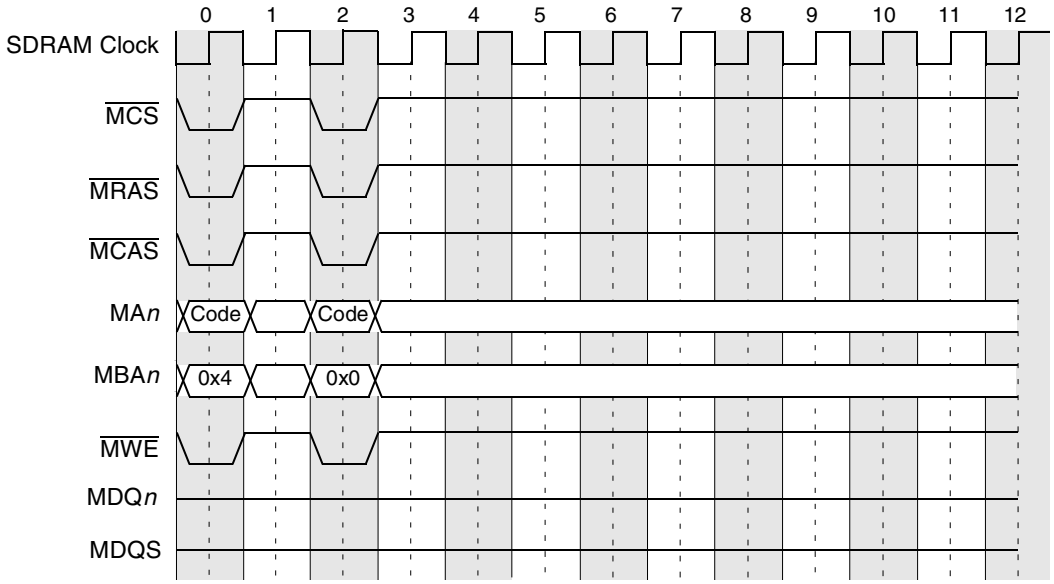


Figure 8-45. DDR SDRAM Mode-Set Command Timing

8.5.6 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 8-46 shows the registered DDR SDRAM DIMM single-beat write timing.

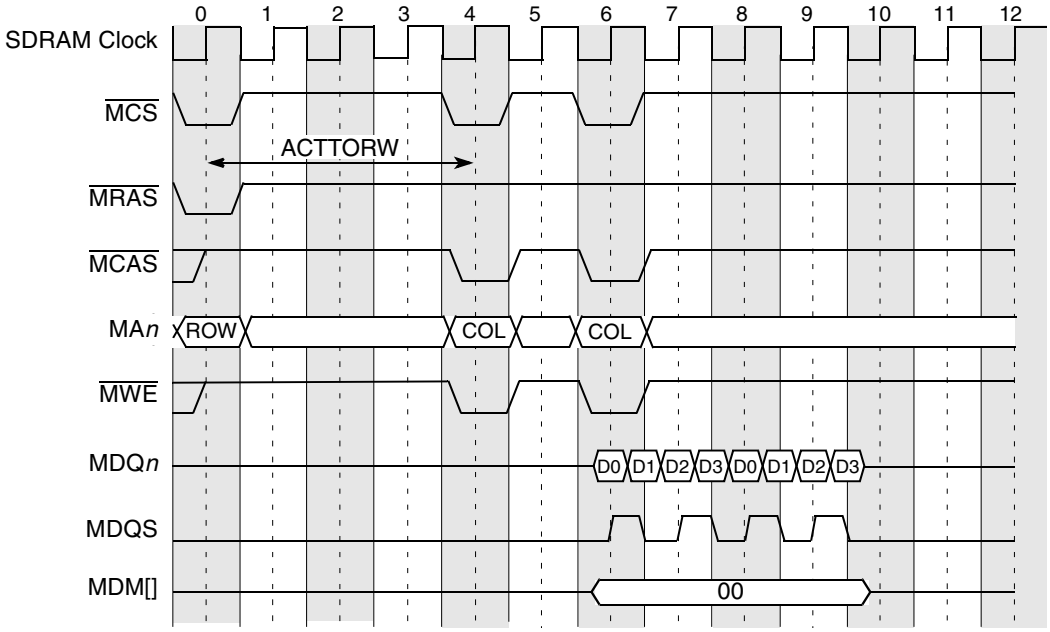


Figure 8-46. Registered DDR SDRAM DIMM Burst Write Timing

8.5.7 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 8-47 shows the use of the WR_DATA_DELAY parameter.

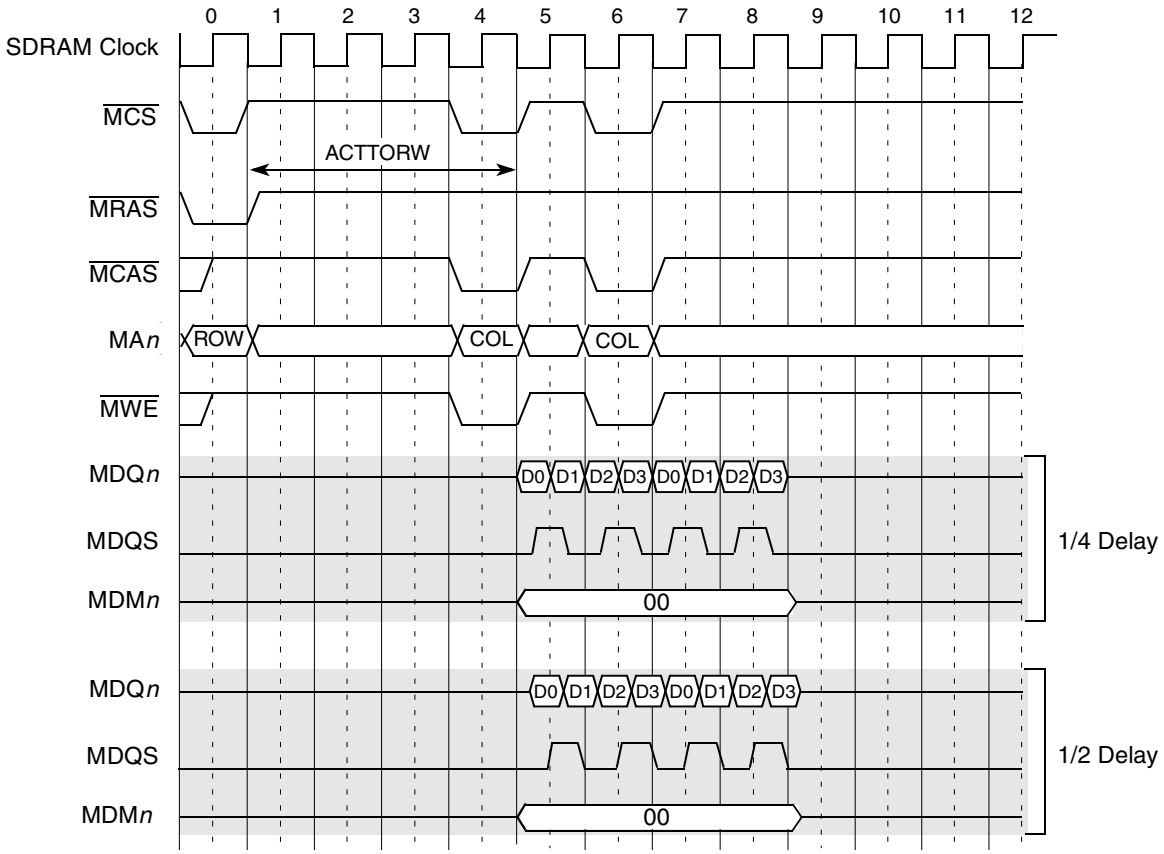


Figure 8-47. Write Timing Adjustments Example for Write Latency = 1

8.5.8 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, must 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.

8.5.8.1 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 8-48 (TIMING_CFG_1 [REFREC] = 10 in this example).

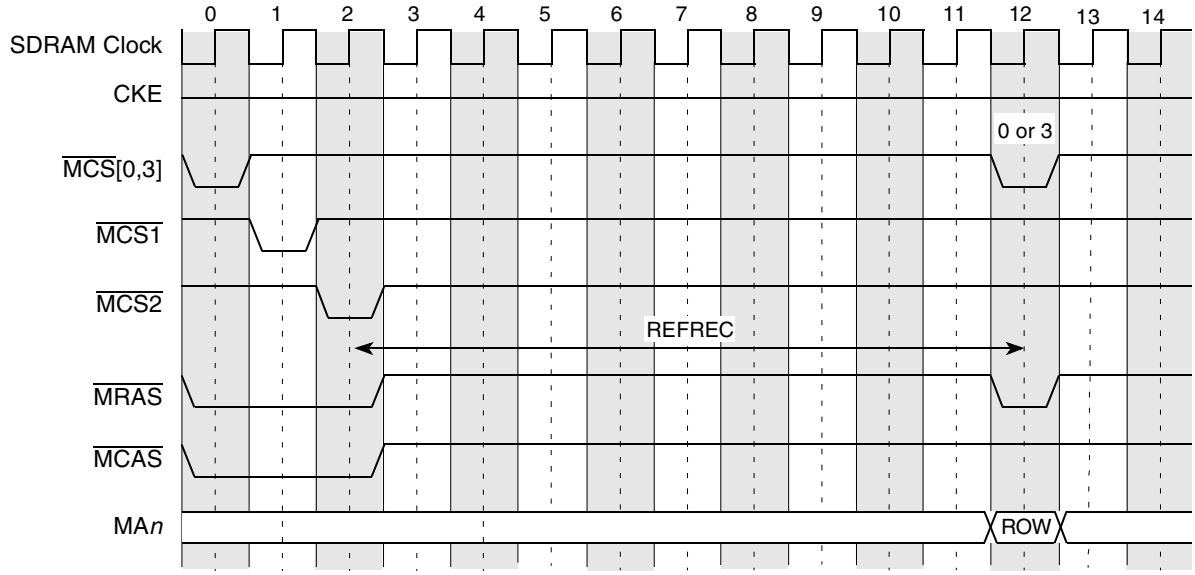


Figure 8-48. DDR SDRAM Bank Staggered Auto Refresh Timing

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.

8.5.8.2 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.

Table 8-55 summarizes the refresh types available in each power-saving mode.

Table 8-55. DDR SDRAM Power-Saving Modes Refresh Configuration

Power Saving Mode	Refresh Type	SREN
Sleep	Self	1
	None	—

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 8-49.

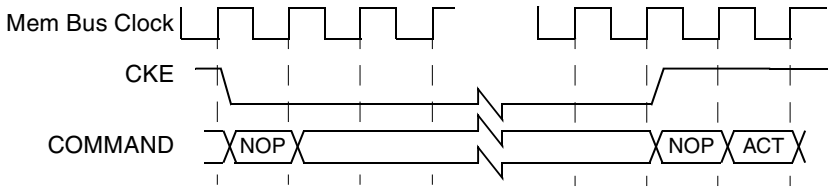


Figure 8-49. DDR SDRAM Power-Down Mode

8.5.8.2.1 Self-Refresh in Sleep Mode

The entry and exit timing for self-refreshing SDRAMs is shown in Figure 8-50 and Figure 8-51.

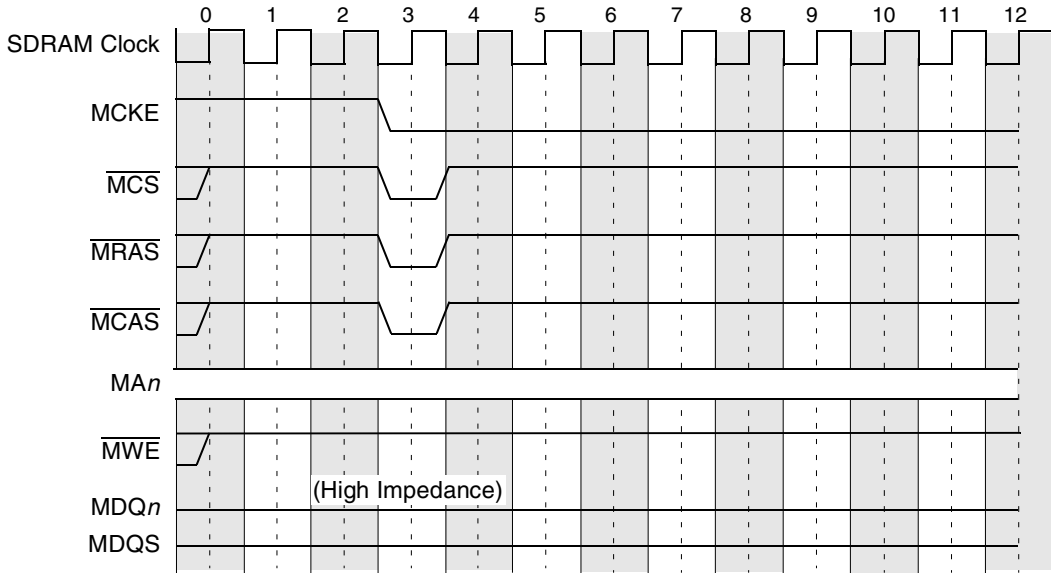


Figure 8-50. DDR SDRAM Self-Refresh Entry Timing

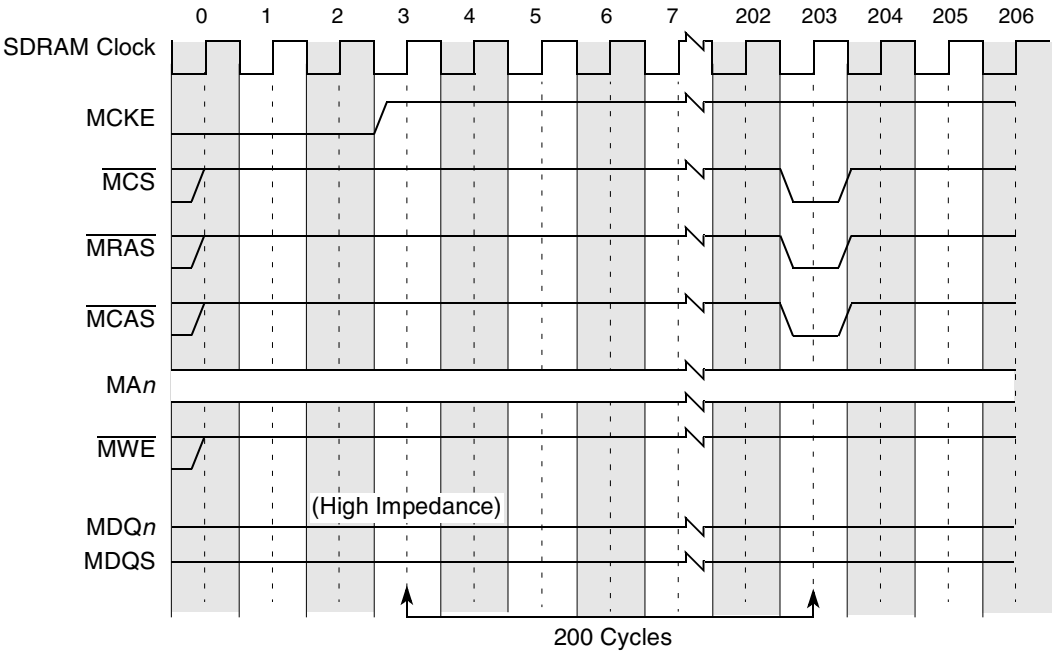


Figure 8-51. DDR SDRAM Self-Refresh Exit Timing

8.5.9 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 double-word aligned or the size is not a multiple of a double word, a full read-modify-write is performed for a write to SDRAM. If ECC is disabled or both the access is double-word aligned with a size that is a multiple of a double word, the data masks MDM[] (MDM[0:] 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, as the width of the data bus is 64 bits.

Table 8-56 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 8-56. 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.

8.5.10 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 must 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].

8.5.11 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. Multi-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 8-57 and Table 8-58.

In 32-bit mode, Table 8-57 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 8-57. DDR SDRAM ECC Syndrome Encoding

Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
0	•	•						•
1	•		•					•
2	•			•				•
3	•				•			•
4	•	•				•		
5	•		•			•		
6	•			•		•		
7	•				•	•		

Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
32			•	•				•
33			•		•			•
34	•		•		•			
35		•	•		•			
36			•	•		•		
37			•		•	•		
38	•		•		•	•		•
39		•	•		•	•		•

Table 8-57. DDR SDRAM ECC Syndrome Encoding (continued)

Data Bit	Syndrome Bit							
	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	•	•			•	•	•	

Data Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
40			•	•			•	
41			•		•		•	
42	•		•		•		•	•
43		•	•		•		•	•
44			•	•		•	•	•
45			•		•	•	•	•
46	•		•		•	•	•	
47		•	•		•	•	•	
48		•				•	•	
49			•			•	•	
50				•		•	•	
51	•					•	•	
52		•				•		•
53			•			•		•
54				•		•		•
55	•					•		•
56		•					•	•
57			•				•	•
58				•			•	•
59	•						•	•
60				•	•		•	
61	•			•	•		•	•
62		•		•	•		•	•
63			•	•	•		•	•

Table 8-58. 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						•		

Table 8-58. DDR SDRAM ECC Syndrome Encoding (Check Bits) (continued)

Check Bit	Syndrome Bit							
	0	1	2	3	4	5	6	7
6							•	
7								•

8.5.12 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 8.4.1.31, “Memory Error Interrupt Enable (ERR_INT_EN),” Section 8.4.1.30, “Memory Error Disable (ERR_DISABLE),” and Section 8.4.1.29, “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, and transfer error acknowledge (TEA) is asserted internally on the CSB bus (if enabled, as described in Section 8.4.1.30, “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 8.4.1.29, “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 8-59 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 8-59. 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.		

8.6 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 $\overline{\text{MCS}}n$ 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 8.4.1.2, "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 8.4.1, "Register Descriptions," for more detailed descriptions of the configuration registers. These parameters are shown in Table 8-60.

Table 8-60. Memory Interface Configuration Register Initialization Parameters

Name	Description	Parameter		Section/page
CS*n*_BNDS	Chip select memory bounds	SA*n* EA*n*		8.4.1.1/8-11
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*	8.4.1.2/8-11
TIMING_CFG_3	Extended timing parameters for fields in TIMING_CFG_1	EXT_REFREC		8.4.1.3/8-13
TIMING_CFG_0	Timing configuration	RWT WRT RRT WWT	ACT_PD_EXIT PRE_PD_EXIT ODT_PD_EXIT MRS_CYC	8.4.1.4/8-14
TIMING_CFG_1	Timing configuration	PRETOACT ACTTOPRE ACTTORW CASLAT	REFREC WRREC ACTTOACT WRTORD	8.4.1.5/8-16
TIMING_CFG_2	Timing configuration	ADD_LAT CPO WR_LAT RD_TO_PRE	WR_DATA_DELAY CKE_PLS FOUR_ACT	8.4.1.6/8-18

Table 8-60. 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	8.4.1.7/8-20
DDR_SDRAM_CFG_2	Control configuration	DLL_RST_DIS DQS_CFG ODT_CFG NUM_PR D_INIT	8.4.1.8/8-23
DDR_SDRAM_MODE	Mode configuration	ESDMODE SDMODE	8.4.1.9/8-24
DDR_SDRAM_MODE_2	Mode configuration	ESDMODE2 ESDMODE3	8.4.1.10/8-25
DDR_SDRAM_INTERVAL	Interval configuration	REFINT BSTOPRE	8.4.1.12/8-28
DDR_DATA_INIT	Data initialization configuration register	INIT_VALUE	8.4.1.13/8-29
DDR_SDRAM_CLK_CNTL	Clock adjust	CLK_ADJUST	8.4.1.14/8-29
DDR_INIT_ADDR	Initialization address	INIT_ADDR	8.4.1.15/8-30
DDR_INIT_EXT_ADDR	Extended initialization address	INIT_EXT_ADDR	8.4.1.16/8-30

8.6.1 Programming Differences between Memory Types

Depending on the memory type used, certain fields must be programmed differently. Table 8-61 illustrates the differences in certain fields for different memory types. Note that this table does not list all fields that must be programmed.

Table 8-61. Programming Differences between Memory Types

Parameter	Description	Differences		Section/page
AP*n*_EN	Chip Select *n* Auto Precharge Enable	DDR1DDR2	Can be used to place chip select *n* in auto precharge mode	8.4.1.2/8-11
ODT_RD_CFG	Chip Select ODT Read Configuration	DDR1	Should always be set to 000	8.4.1.2/8-11
		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 8-61. 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	8.4.1.2/8-11
		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	8.4.1.4/8-14
		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})	8.4.1.5/8-16
		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})	8.4.1.5/8-16
		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})	8.4.1.5/8-16
		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	8.4.1.5/8-16
		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})	8.4.1.5/8-16
		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})	8.4.1.5/8-16
		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})	8.4.1.5/8-16
		DDR2	Should be set according to the specifications for the memory used (t_{RRD})	

Table 8-61. 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})	8.4.1.5/8-16
		DDR2	Should be set according to the specifications for the memory used (t_{WTR})	
ADD_LAT	Additive Latency	DDR1	Should be set to 000	8.4.1.6/8-18
		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	8.4.1.6/8-18
		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	8.4.1.6/8-18
		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	8.4.1.6/8-18
		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	8.4.1.6/8-18
		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	8.4.1.7/8-20
		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	8.4.1.7/8-20
		DDR2	Should be set to 0 regardless of bus width. In 32-bit bus mode, 32-byte burst accesses from the platform are split into two 16-byte (4-beat) bursts to the SDRAMs.	
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.	8.4.1.7/8-20
		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 8-61. 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.	8.4.1.8/8-23
		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	8.4.1.8/8-23
		DDR2	Can be set to either 00 or 01, depending on if differential strobes are used	
ODT_CFG	ODT Configuration	DDR1	Should be set to 00	8.4.1.8/8-23
		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.	8.4.1.12/8-28
		DDR2	Can be set to any value, depending on the application. Auto precharge can be enabled by setting this field to all 0s.	

8.6.2 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.

Chapter 9
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.

9.1 Introduction

Figure 9-1 is a functional block diagram of the eLBC, which supports three interfaces: GPCM, FCM, and UPM controllers.

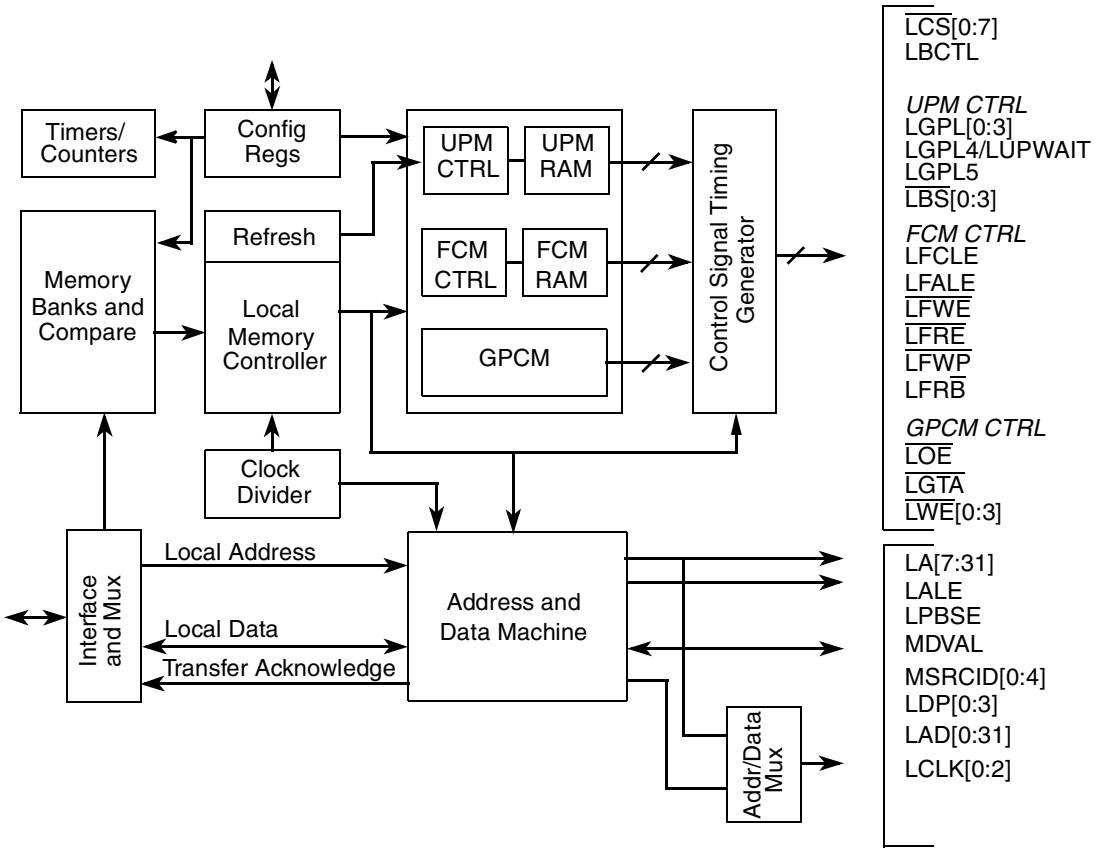


Figure 9-1. Enhanced Local Bus Controller Block Diagram

9.1.1 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.

9.1.2 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
  - Atomic operation
  - 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
  - ECC checking enable/disable feature supported during boot

  - 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)
- Different machines (FCM/GPCM/UPM) share the address, data, and control signals. While the eLBC is servicing a transaction, subsequent transactions are queued until the current transaction has completed.

9.1.3 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.

9.1.3.1 eLBC Bus Clock and Clock Ratios

The eLBC supports ratios of 4, 8, and 16 between the faster internal (platform) 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.

9.1.3.2 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 MSRCID[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 (MDVAL) facilitate capturing useful debug data as follows:

- If a valid source ID is detected on MSRCID[0:4] and LALE is asserted, a valid full 26-bit address may be latched from LAD[6:26] and combined with any LA[6:31] or when AS16=1, LAD[0:15] and combined with LA[22:31].
- If a valid source ID is detected on MSRCID[0:4] and MDVAL is asserted, valid data may be latched from LAD.

The MSRCID[0:4] and MDVAL signals are multiplexed with other functions sharing the same external pins. Refer to Chapter 4, “Reset, Clocking, and Initialization,” to learn how to enable the MSRCID/MDVAL pins.

9.2 External Signal Descriptions

Table 9-1 contains a list of external signals related to the eLBC and summarizes their function.The table also shows the reset state of all external signals during assertion of $\overline{\text{HRESET}}$. For more information on the use of some of these signals as reset configuration signals on the device, see “Power on Reset Flow.”

Table 9-1. 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		

Table 9-1. Signal Properties—Summary (continued)

Name	Alternate Function(s)	Mode	Descriptions	No. of Signals	I/O	Reset State (Outputs)
$\overline{\text{LWE}}$[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		—
LGPL1/ LFALE	LGPL1	UPM	General purpose line 1	1	O	Reset_cfg
	LFALE	FCM	Flash address latch enable	1		—
$\overline{\text{LOE}}$/ LGPL2/ $\overline{\text{LFRE}}$	$\overline{\text{LOE}}$	GPCM	Output enable	1	O	—
	$\overline{\text{LFRE}}$	FCM	Flash read enable	1		—
	LGPL2	UPM	General purpose line 2	1		—
LGPL3/ $\overline{\text{LFWP}}$	LGPL3	UPM	General purpose line 3	1	O	Reset_cfg
	$\overline{\text{LFWP}}$	FCM	Flash write protect	1		—
$\overline{\text{LGTA}}$/ LFR$\overline{\text{B}}$/ LGPL4/ LUPWAIT/ LPBSE	$\overline{\text{LGTA}}$	GPCM	Transaction termination	1	I	High-Z
	LFR$\overline{\text{B}}$	FCM	Flash ready/$\overline{\text{busy}}$, 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
MDVAL	—	eLBC debug	Local bus data valid	1	O	
MSRCID[0:4]	—	eLBC debug	Local bus source ID	5	O	

Table 9-2 shows the detailed external signal descriptions for the eLBC.

Table 9-2. 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 9-2. 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 9.4.4.4, "RAM Array," for programming details about LBS[0:3].
		Timing	Assertion/Negation—See Section 9.4.2, "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 9.3.1.16, "Flash Mode Register (FMR)," for programming of FCM operations to control LFWP.

Table 9-2. 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*n* 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 9.5, "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 9-2. 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.
MDVAL	O	Local bus data valid (eLBC debug mode only)	
		State Meaning	Asserted/Negated—For a read, MDVAL asserts for one bus cycle in the cycle immediately preceding the sampling of read data on LAD. For a write, MDVAL asserts for one bus cycle during the final cycle for which the current write data on LAD is valid. During burst transfers, MDVAL asserts for each data beat.
		Timing	Assertion/Negation—Valid only while the eLBC is in system debug mode. In debug mode, MDVAL asserts when the eLBC generates a data transfer acknowledge.
MSRCID[0:4]	O	Local bus source ID (eLBC debug mode only). In debug mode, all MSRCID[0:4] pins are driven high unless MSRCID[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 MDVAL is asserted, in which case the source ID relating to the data transfer is indicated. In case of address debug, MSRCID[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.

9.3 Memory Map/Register Definition

Table 9-3 shows the memory mapped registers of the eLBC. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 9-3. 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*	9.3.1.1/9-10
0x008	BR1—Base register 1	R/W	All zeros	9.3.1.1/9-10
0x010	BR2—Base register 2	R/W	All zeros	9.3.1.1/9-10

Table 9-3. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x018	BR3—Base register 3	R/W	All zeros	9.3.1.1/9-10
0x020	BR4—Base register 4	R/W	All zeros	9.3.1.1/9-10
0x028	BR5—Base register 5	R/W	All zeros	9.3.1.1/9-10
0x030	BR6—Base register 6	R/W	All zeros	9.3.1.1/9-10
0x038	BR7—Base register 7	R/W	All zeros	9.3.1.1/9-10
0x004	OR0—Options register 0	R/W	0x0000_0FF7	9.3.1.2/9-12
0x00C	OR1—Options register 1	R/W	All zeros	9.3.1.2/9-12
0x014	OR2—Options register 2	R/W	All zeros	9.3.1.2/9-12
0x01C	OR3—Options register 3	R/W	All zeros	9.3.1.2/9-12
0x024	OR4—Options register 4	R/W	All zeros	9.3.1.2/9-12
0x02C	OR5—Options register 5	R/W	All zeros	9.3.1.2/9-12
0x034	OR6—Options register 6	R/W	All zeros	9.3.1.2/9-12
0x03C	OR7—Options register 7	R/W	All zeros	9.3.1.2/9-12
0x040–0x064	Reserved	—	—	—
0x068	MAR—UPM address register	R/W	All zeros	9.3.1.3/9-20
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	All zeros	9.3.1.4/9-21
0x074	MBMR—UPMB mode register	R/W	All zeros	9.3.1.4/9-21
0x078	MCMR—UPMC mode register	R/W	All zeros	9.3.1.4/9-21
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	All zeros	9.3.1.5/9-23
0x088	MDR—UPM/FCM data register	R/W	All zeros	9.3.1.6/9-23
0x08C	Reserved	—	—	—
0x090	LSOR—Special operation initiation register	R/W	All zeros	9.3.1.7/9-24
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	All zeros	9.3.1.8/9-25
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	All zeros	9.3.1.9/9-26
0x0B4	LTEDR—Transfer error disable register	R/W	All zeros	9.3.1.10/9-28
0x0B8	LTEIR—Transfer error interrupt register	R/W	All zeros	9.3.1.11/9-29
0x0BC	LTEATR—Transfer error attributes register	R/W	All zeros	9.3.1.12/9-30

Table 9-3. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0C0	LTEAR—Transfer error address register	R/W	All zeros	9.3.1.13/9-31
0x0C4–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W	All zeros	9.3.1.14/9-32
0x0D4	LCRR—Clock ratio register	R/W	0x8000_0008	9.3.1.15/9-33
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0*n*00	9.3.1.16/9-34
0x0E4	FIR—Flash instruction register	R/W	All zeros	9.3.1.17/9-36
0x0E8	FCR—Flash command register	R/W	All zeros	9.3.1.18/9-37
0x0EC	FBAR—Flash block address register	R/W	All zeros	9.3.1.19/9-38
0x0F0	FPAR—Flash page address register	R/W	All zeros	9.3.1.20/9-38
0x0F4	FBCR—Flash byte count register	R/W	All zeros	9.3.1.21/9-40
0x0F8–0x0FC	Reserved	—	—	—

9.3.1 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 9-3 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.

9.3.1.1 Base Registers (BR0–BR7)

The base registers (BR*n*), shown in Figure 9-2, 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 power-on configuration settings.

Offset BR0: 0x000 BR4: 0x020
BR1: 0x008 BR5: 0x028
BR2: 0x010 BR6: 0x030
BR3: 0x018 BR7: 0x038

Access: Read/Write

	0															16	17	18	19	20	21	22	23	24		26	27	28	29	30	31
R W	BA															—			PS		DECC		WP	MSEL			—	ATOM		—	V
Reset	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	0	V[1]

[1] BR0 has its valid bit (V) set at reset. Thus bank 0 is valid with the port size (PS) configured from cfg_rom_loc[0:3] at power-on reset. M = 0 for MSEL of GPCM, 1 for MSEL of FCM at boot. The reset value for DECC is determined by cfg_elbc_ecc. All other base registers have all bits cleared to zero during reset.

Figure 9-2. Base Registers (BR*n*)

Table 9-4 describes BR*n* fields.

Table 9-4. 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 boot ROM location power-on reset configuration setting. For all other banks the value is reset to 00 (port size not defined). 00 Reserved 01 8-bit (supported for GPCM, UPM, FCM) 10 16-bit (supported for GPCM, UPM, 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
23	WP	Write protect. 0 Read and write accesses are allowed. 1 Only read accesses are allowed. The memory controller does not assert $\overline{\text{LCS}n}$ 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.

Table 9-4. BR*n* Field Descriptions (continued)

Bits	Name	Description
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	ATOM	Atomic operation. Writes (reads) to the address space handled by the memory controller bank reserve the selected memory bank for the exclusive use of the accessing device. The reservation is released when the device performs a read (write) operation to this memory controller bank. If a subsequent read (write) request to this memory controller bank is not detected within 256 bus clock cycles of the last write (read), the reservation is released and an atomic error is reported (if enabled). 00 The address space controlled by this bank is not used for atomic operations. 01 Read-after-write-atomic (RAWA). 10 Write-after-read-atomic (WARA). 11 Reserved
30	—	Reserved
31	V	Valid bit. Indicates that the contents of the BR*n* and OR*n* pair are valid. $\overline{\text{LCS}n}$ 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.

9.3.1.2 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 9-5 shows the reset values for OR0.

Table 9-5. 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

9.3.1.2.1 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 9-6 shows memory bank sizes from 32 Kbytes to 4 Gbytes.

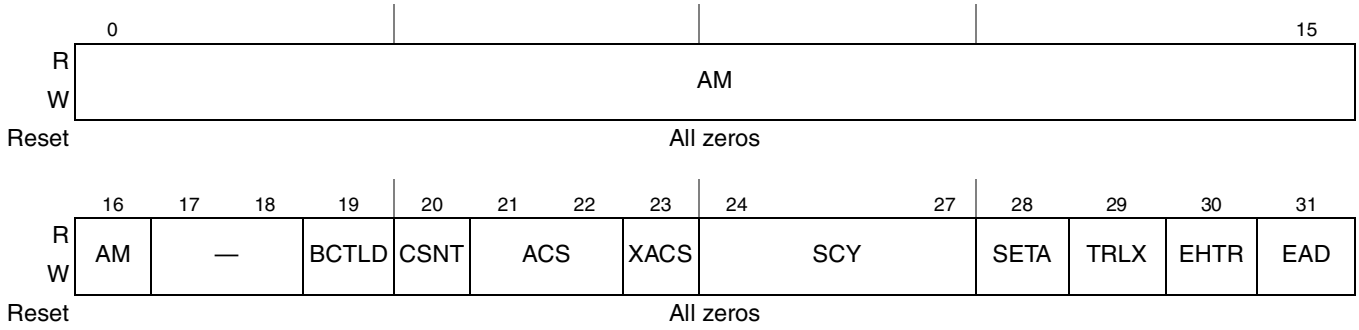
Table 9-6. Memory Bank Sizes in Relation to Address Mask

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

9.3.1.2.2 Option Registers (OR*n*)—GPCM Mode

Figure 9-3 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the GPCM machine.

Offset OR0: 0x004 OR4: 0x024 Access: Read/Write
OR1: 0x00C OR5: 0x02C
OR2: 0x014 OR6: 0x034
OR3: 0x01C OR7: 0x03C



1 Refer to Table 9-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 9-3. Option Registers (OR*n*) in GPCM Mode

Table 9-7 describes OR*n* fields for GPCM mode.

Table 9-7. 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.
20	CSNT	Chip select negation time. Determines when $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated during an external memory write access handled by the GPCM, provided that ACS ≠ 00 (when ACS = 00, only $\overline{\text{LWE}}$ is affected by the setting of CSNT). This helps meet address/data hold times for slow memories and peripherals. 0 $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated normally. 1 $\overline{\text{LCS}n}$ and $\overline{\text{LWE}}$ are negated one quarter of a bus clock cycle earlier.
21–22	ACS	Address to chip-select setup. Determines the delay of the $\overline{\text{LCS}n}$ assertion relative to the address change when the external memory access is handled by the GPCM. At system reset, OR0[ACS] = 11. 00 $\overline{\text{LCS}n}$ is output at the same time as the address lines. Note that this overrides the value of CSNT such that CSNT = 0. 01 Reserved. 10 $\overline{\text{LCS}n}$ is output one quarter bus clock cycle after the address lines. 11 $\overline{\text{LCS}n}$ is output one half bus clock cycle after the address lines.

Table 9-7. OR*n*—GPCM Field Descriptions (continued)

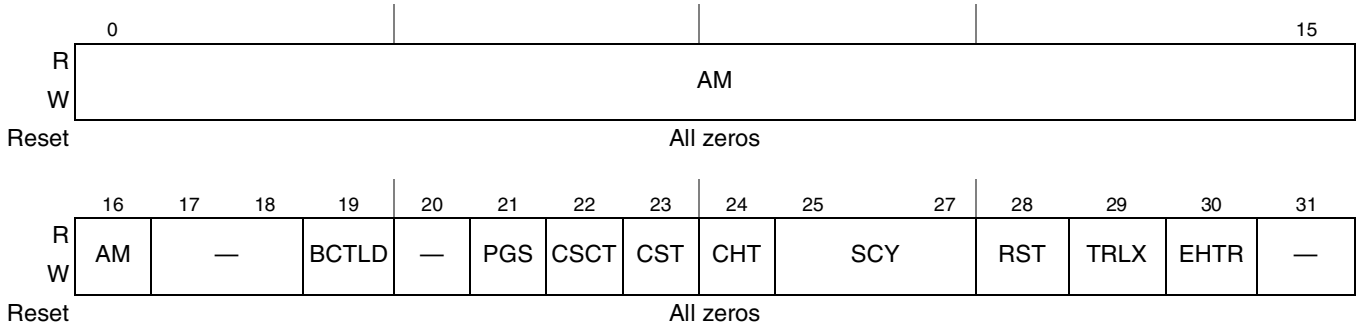
Bits	Name	Description
23	XACS	Extra address to chip-select setup. Setting this bit increases the delay of the $\overline{\text{LCS}n}$ assertion relative to the address change when the external memory access is handled by the GPCM. After a system reset, OR0[XACS] = 1. 0 Address to chip-select setup is determined by ORx[ACS]. 1 Address to chip-select setup is extended (see Table 9-30 and Table 9-31).
24–27	SCY	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
28	SETA	External address termination. 0 Access is terminated internally by the memory controller unless the external device asserts $\overline{\text{LGTA}}$ earlier to terminate the access. 1 Access is terminated externally by asserting the $\overline{\text{LGTA}}$ external pin. (Only $\overline{\text{LGTA}}$ can terminate the access).
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.
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]).

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.

9.3.1.2.3 Option Registers (OR*n*)—FCM Mode

Figure 9-4 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects the FCM machine.

Offset OR0: 0x004 OR4: 0x024 Access: Read/Write
OR1: 0x00C OR5: 0x02C
OR2: 0x014 OR6: 0x034
OR3: 0x01C OR7: 0x03C



1 Refer to Table 9-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 9-4. Option Registers (OR*n*) in FCM Mode

Table 9-8 describes OR*n* fields for FCM mode.

Table 9-8. OR*n*—FCM Field Descriptions

Bits	Name	Description
0–16	AM	FCM 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. 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.
20	—	Reserved
21	PGS	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.

Table 9-8. OR*n*—FCM Field Descriptions (continued)

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></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>
<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>
</table>

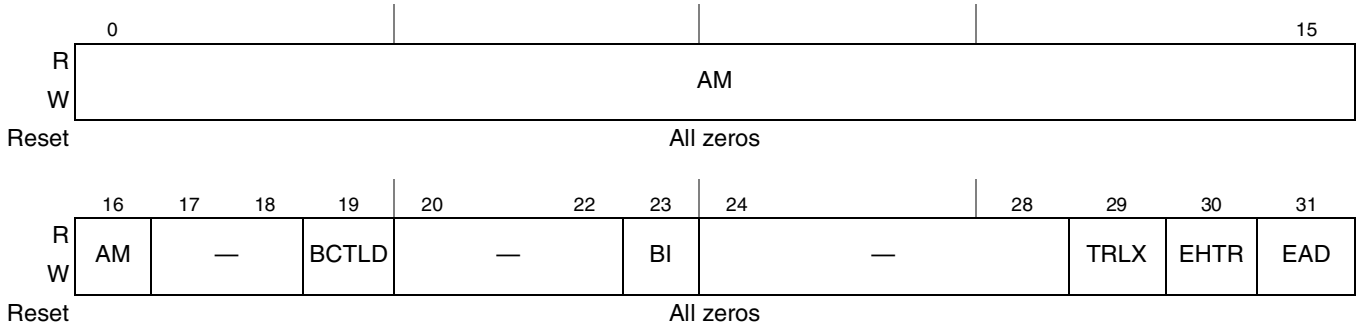
Table 9-8. OR*n*—FCM Field Descriptions (continued)

Bits	Name	Description
25–27	SCY	Cycle length in bus clocks. Determines the following: • 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
28	RST	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. **TRLX / RST / Meaning** 0 / 0 / The read-enable is asserted 0.75 clock cycles prior to any wait states. 0 / 1 / The read-enable is asserted 1 clock cycle prior to any wait states. 1 / 0 / The read-enable is asserted 0.5 clock cycles prior to any wait states. 1 / 1 / The read-enable is asserted 1 clock cycle prior to any wait states.
29	TRLX	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.
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 / 1 idle clock cycle is inserted. 0 / 1 / 2 idle clock cycles are inserted. 1 / 0 / 4 idle clock cycles are inserted. 1 / 1 / 8 idle clock cycles are inserted.
31	—	Reserved

9.3.1.2.4 Option Registers (OR*n*)—UPM Mode

Figure 9-5 shows the bit fields for OR*n* when the corresponding BR*n*[MSEL] selects a UPM machine.

Offset OR0: 0x004 OR4: 0x024 Access: Read/Write
OR1: 0x00C OR5: 0x02C
OR2: 0x014 OR6: 0x034
OR3: 0x01C OR7: 0x03C



1 Refer to Table 9-5 for the OR0 reset value. All other option registers have all bits cleared.

Figure 9-5. Option Registers (OR*n*) in UPM Mode

Table 9-9 describes BR*n* fields for UPM mode.

Table 9-9. OR*n*—UPM Field Descriptions

Bits	Name	Description
0–16	AM	UPM 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. 1 The corresponding address bits are used in the comparison with address pins.
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.
20–22	—	Reserved
23	BI	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.
24–28	—	Reserved
29	TRLX	Timing relaxed. Works in conjunction with EHTR to extend hold time on read accesses.

Table 9-9. OR*n*—UPM Field Descriptions (continued)

Bits	Name	Description
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]).

9.3.1.3 UPM Memory Address Register (MAR)

Figure 9-6 shows the fields of the UPM memory address register (MAR).

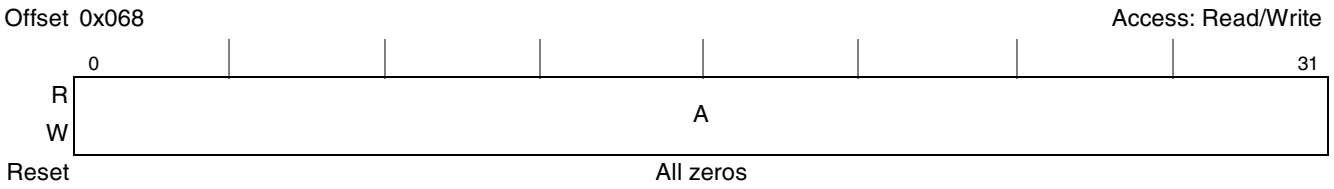


Figure 9-6. UPM Memory Address Register (MAR)

Table 9-10 describes the MAR fields.

Table 9-10. 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.

9.3.1.4 UPM Mode Registers (M*x*MR)

The UPM machine mode registers (MAMR, MBMR, and MCMR), shown in Figure 9-7, contain the configuration for the three UPMs.

Offset MAMR: 0x070
MBMR: 0x074
MCMR: 0x078

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	—	RFEN	OP		UWPL	AM			DS		G0CL			GPL4	RLF	
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	RLF		WLF				TLF				MAD					
Reset	All zeros															

Figure 9-7. UPM Mode Registers (M*x*MR)

Table 9-11 describes UPM mode fields.

Table 9-11. 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 9-11. M*x*MR 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 9.4.4.4.7, “Address Multiplexing (AMX)” for more information.
000 Internal transaction address a[8:23] driven on LA/LAD[16:31]; LAD[0:15] driven low.
001 Internal transaction address a[7:22] driven on LA/LAD[16:31]; LAD[0:15] driven low.
010 Internal transaction address a[6:21] driven on LA/LAD[16:31]; LAD[0:15] driven low.
011 Internal transaction address a[5:20] driven on LA/LAD[16:31]; LAD[0:15] driven low.
100 Internal transaction address a[4:19] driven on LA/LAD[16:31]; LAD[0:15] driven low.
101 Internal transaction address a[3:18] driven on LA/LAD[16:31]; 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 9-38.
<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 9-11. M*x*MR 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*.

9.3.1.5 Memory Refresh Timer Prescaler Register (MRTPR)

The refresh timer prescaler register (MRTPR), shown in Figure 9-8, is used to divide the platform clock to provide the UPM refresh timers clock.

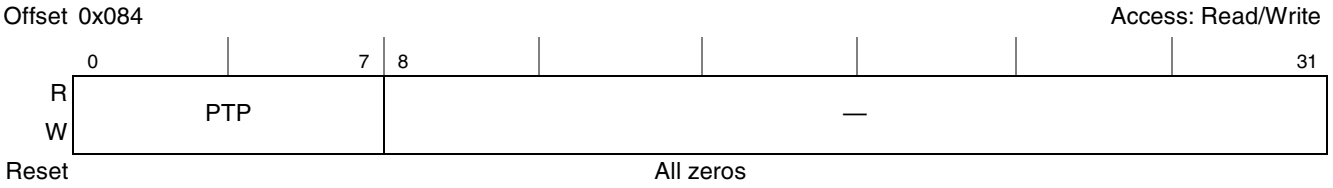


Figure 9-8. Memory Refresh Timer Prescaler Register (MRTPR)

Table 9-12 describes MRTPR fields.

Table 9-12. MRTPR Field Descriptions

Bits	Name	Description
0–7	PTP	Refresh timers prescaler. Determines the period of the refresh timers input clock. The platform clock is divided by PTP except when the value is 00000_0000, which represents the maximum divider of 256.
8–31	—	Reserved

9.3.1.6 UPM/FCM Data Register (MDR)

The memory data register (MDR), shown in Figure 9-9 and Figure 9-10, 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.

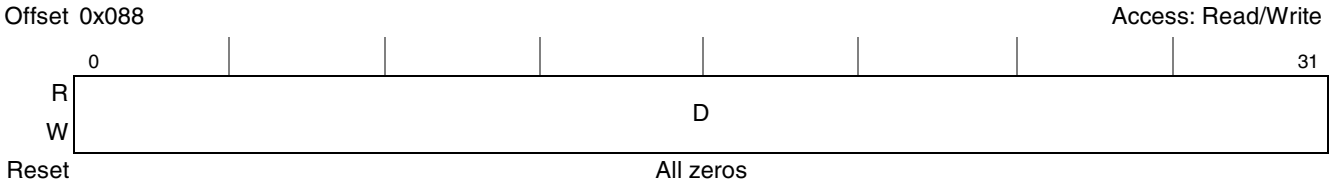


Figure 9-9. UPM Data Register in UPM Mode (MDR)

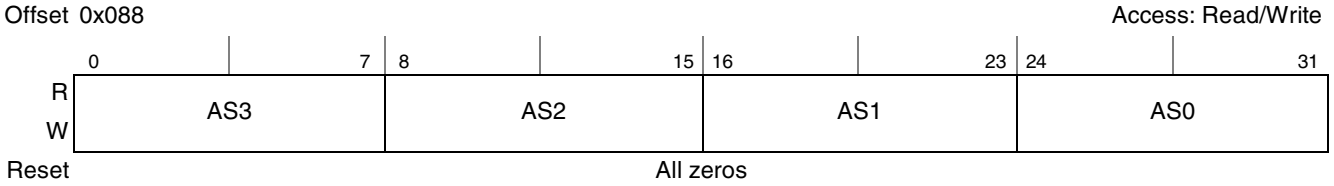


Figure 9-10. FCM Data Register in FCM Mode (MDR)

Table 9-13 describes MDR[D].

Table 9-13. 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.

9.3.1.7 Special Operation Initiation Register (LSOR)

The special operation initiation register (LSOR), shown in Figure 9-11, 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 rewritten 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 9.3.1.4, "UPM Mode Registers (MxMR)." FCM special operation modes are set in FMR[OP], see Section 9.3.1.16, "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 9.4.4.2.1, "UPM Programming Example (Two Sequential Writes to the RAM Array)."

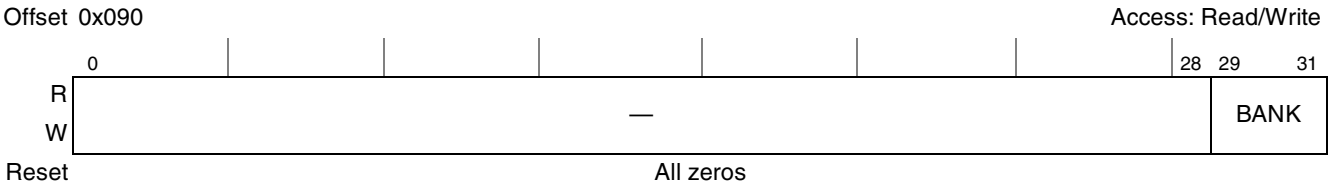


Figure 9-11. Special Operation Initiation Register (LSOR)

Table 9-14 describes LSOR.

Table 9-14. 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

9.3.1.8 UPM Refresh Timer (LURT)

The UPM refresh timer (LURT), shown in Figure 9-12, 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.

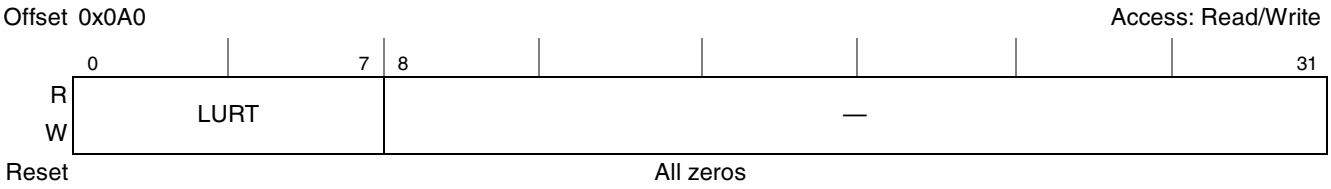


Figure 9-12. UPM Refresh Timer (LURT)

Table 9-15 describes LURT fields.

Table 9-15. 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: $$TimerPeriod = \frac{LURT}{\left(\frac{Fsystemclock}{MRTPR[PTP]}\right)}$$ 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

9.3.1.9 Transfer Error Status Register (LTESR)

The transfer error status register (LTESR) indicates the cause of an error or event. LTESR, shown in Figure 9-13, 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.

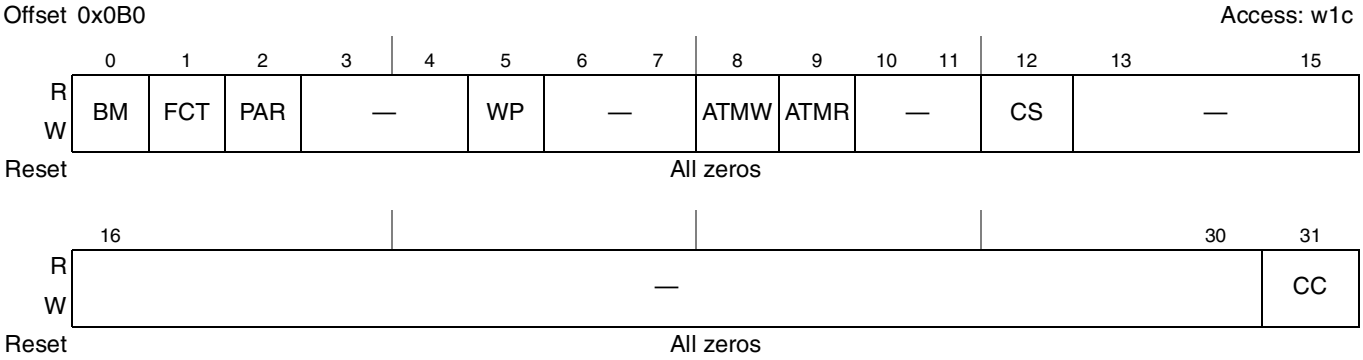


Figure 9-13. Transfer Error Status Register (LTESR)

Table 9-16 describes LTESR fields.

Table 9-16. 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 non-correctable 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).
6–7	—	Reserved
8	ATMW	Atomic error write 0 No atomic write error occurred. 1 The subsequent write (WARA) to a memory bank did not occur within 256 bus clock cycles.
9	ATMR	Atomic error read 0 No atomic read error occurred. 1 The subsequent read (RAWA) to a memory bank did not occur within 256 bus clock cycles.
10–11	—	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.

9.3.1.10 Transfer Error Check Disable Register (LTEDR)

The transfer error check disable register (LTEDR), shown in Figure 9-14, 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 0x0B4 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	15
R W	BMD	FCTD	PARD	—		WPD	—		WARA	RAWA	—		CSD	—	
Reset	All zeros														

	16	30	31
R W	—		CCD
Reset	All zeros		

Figure 9-14. Transfer Error Check Disable Register (LTEDR)

Table 9-17 describes LTEDR fields.

Table 9-17. 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.
6–7	—	Reserved
8	WARA	Write after read atomic (WARA) error checking disable. 0 WARA error checking is enabled. 1 WARA error checking is disabled.
9	RAWA	Read after write atomic (RAWA) error checking disable. 0 RAWA error checking is enabled. 1 RAWA error checking is disabled.
10–11	—	Reserved
12	CSD	Chip select error checking disable. 0 Chip select error checking is enabled. 1 Chip select error checking is disabled.

Table 9-17. LTEDR Field Descriptions (continued)

Bits	Name	Description
13–30	—	Reserved
31	CCD	FCM command completion checking disable. 0 Command completion checking is enabled. 1 Command completion checking is disabled.

9.3.1.11 Transfer Error Interrupt Enable Register (LTEIR)

The transfer error interrupt enable register (LTEIR), shown in Figure 9-15, 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.

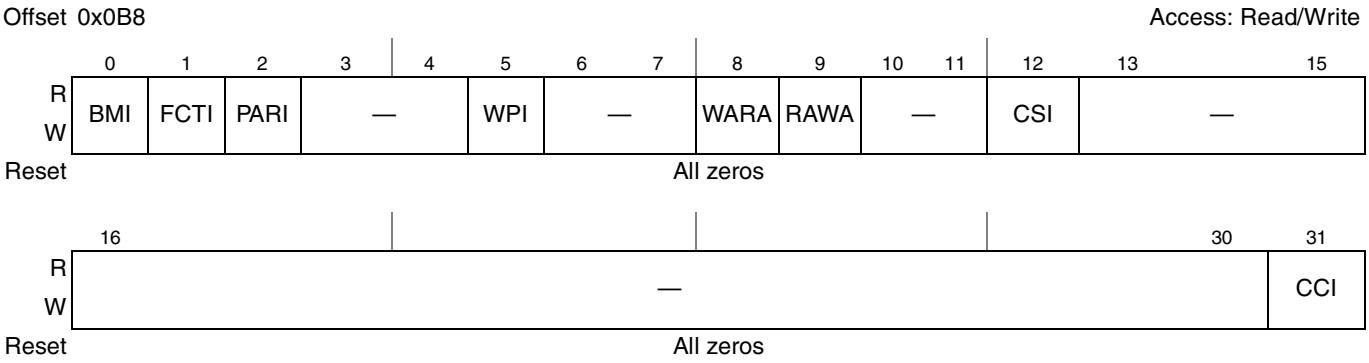


Figure 9-15. Transfer Error Interrupt Enable Register (LTEIR)

Table 9-18 describes LTEIR fields.

Table 9-18. 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.
6–7	—	Reserved

Table 9-18. LTEIR Field Descriptions (continued)

Bits	Name	Description
8	WARA	Write after read atomic (WARA) error interrupt enable. 0 WARA error reporting is disabled. 1 WARA error reporting is enabled.
9	RAWA	Read after write atomic (RAWA) error interrupt enable. 0 RAWA error reporting is disabled. 1 RAWA error reporting is enabled.
10–11	—	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.

9.3.1.12 Transfer Error Attributes Register (LTEATR)

The transfer error attributes register (LTEATR) captures source attributes of an error/event. Figure 9-16 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.

NOTE

LTEATR may not capture accurate information for errors that occur when an FCM special operation is in progress.

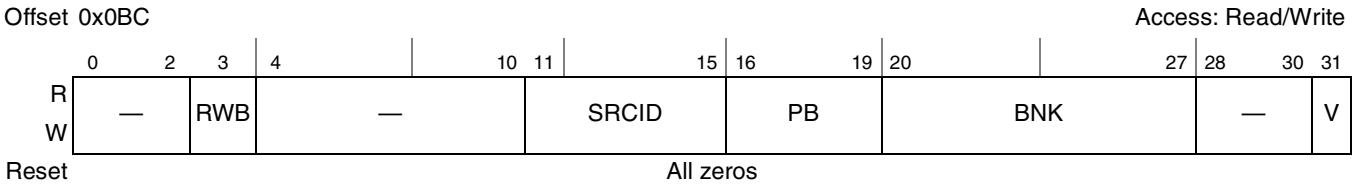


Figure 9-16. Transfer Error Attributes Register (LTEATR

Table 9-19 describes LTEATR fields.

Table 9-19. 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

Table 9-19. LTEATR Field Descriptions (continued)

Bits	Name	Description
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 non-correctable 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.

9.3.1.13 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 9-17.

NOTE

LTEAR may not capture accurate information for errors that occur when an FCM special operation is in progress.

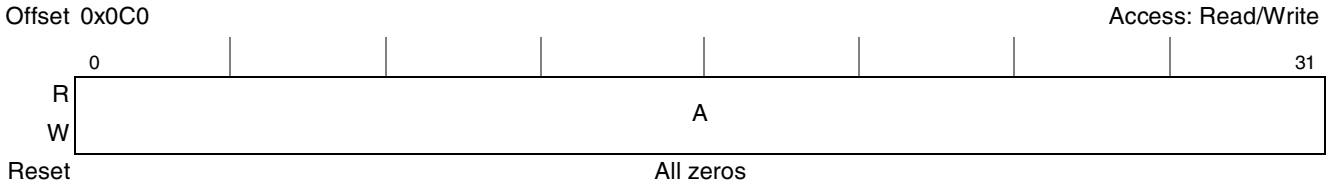


Figure 9-17. Transfer Error Address Register (LTEAR)

Table 9-20 describes LTEAR fields.

Table 9-20. 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.

9.3.1.14 Local Bus Configuration Register (LBCR)

The local bus configuration register (LBCR) is shown in Figure 9-18.

Offset 0x0D0 Access: Read/Write

	0	1–7	8–9	10	11–12	13	14	15
R W	LDIS	—	BCTLC	AHD	—	AS16	LPBSE	EPAR
Reset	All zeros							

	16–23	24–27	28–31
R W	BMT	—	BMTPS
Reset	All zeros		

Figure 9-18. Local Bus Configuration Register

Table 9-21 describes LBCR fields.

Table 9-21. 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/$\overline{R}$ control for GPCM or UPM accesses (buffer control). 01 LBCTL is used as $\overline{LOE}$ for GPCM accesses only. 10 LBCTL is used as $\overline{LWE}$ 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 333 MHz, this provides 3 ns of additional address hold time at the external address latch. Running the platform at a lower frequency improves the hold time. 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.
11–12	—	Reserved
13	AS16	Address shift for 16-bit port size. 0 For all port sizes the most-significant bit of the memory address during LALE assertion appears on the LAD[0] signal, while less-significant bits appear on subsequent signals, with the least-significant bit appearing on line LAD[31]. The 26 least-significant address bits are also driven on LA[6:31] for an entire memory access. 1 Same address bit assignment as AS16 = 0 for 32-bit port sizes. However, for port sizes smaller than 32 bits, the address driven on LA[6:21] is assigned to LAD[0:15] during LALE assertion, while the 10 least-significant address bits are driven on LA[22:31]. Thus the upper 16 bits of the 26-bit memory address may be latched from LAD[0:15].

Table 9-21. LBCR Field Descriptions (continued)

Bits	Name	Description
14	LPBSE	Enables parity byte select on $\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/LGPL4/LUPWAIT/LPBSE signal. 0 Parity byte select is disabled. $\overline{\text{LGTA}}$/LGPL4/LPBSE signal is available for memory control as LGPL4 (output) or $\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/LUPWAIT (input). 1 Parity byte select is enabled. LPBSE signal is dedicated as the parity byte select output, and $\overline{\text{LGTA}}$/LFR$\overline{\text{B}}$/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.
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

9.3.1.15 Clock Ratio Register (LCRR)

The clock ratio register, shown in Figure 9-19, sets the platform 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 0x0D4 Access: Read/Write

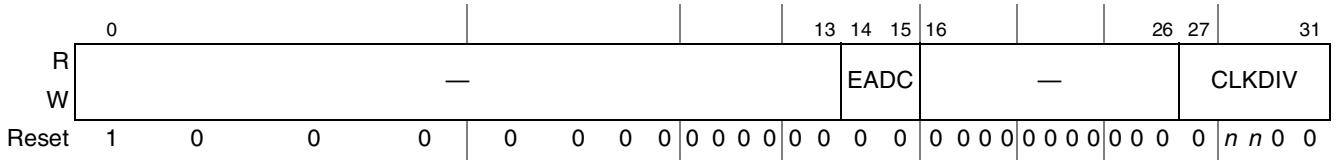


Figure 9-19. Clock Ratio Register (LCRR)

Table 9-22 describes LCRR fields.

Table 9-22. LCRR Field Descriptions

Bits	Name	Description
0–13	—	Reserved. Note bit 0 must remain set for proper operation.
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	Clock divider. Sets the frequency ratio between the platform clock and the local bus clock. 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 4 00011 Reserved 00100 8 00101–00111 Reserved 01000 16 01001–11111 Reserved

9.3.1.16 Flash Mode Register (FMR)

The local bus Flash mode register (FMR), shown in Figure 9-20, controls global operation of the FCM.

Offset 0x0E0 Access: Read/Write

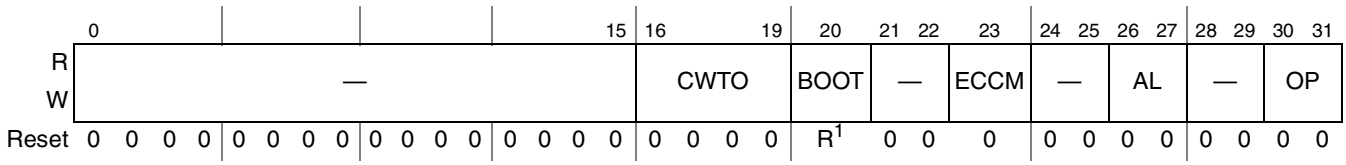


Figure 9-20. Flash Mode Register

1 Bit R (field BOOT) is set if power-on-reset configuration selects FCM as the boot ROM target.

Table 9-23 describes FMR fields.

Table 9-23. FMR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–19	CWTO	Command wait time-out. For FCM commands that wait on LFR$\overline{B}$ being sampled high (CW0, CW1, RBW, and RSW), FCM pauses execution of the instruction sequence until either LFR$\overline{B}$ is sampled high, or a timer controlled by CTO expires, whichever occurs first. The time-out in the latter case is as follows: 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 autoload 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 ($N \times 16$) + 6 through ($N \times 16$) + 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 ($N \times 16$) + 8 through ($N \times 16$) + 10 for spare region N, N = 0–3.
24–25	—	Reserved

Table 9-23. 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 $\overline{\text{LFWP}}$ 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 $\overline{\text{LFWP}}$ remaining high during the access.

9.3.1.17 Flash Instruction Register (FIR)

The local bus Flash instruction register (FIR), shown in Figure 9-21, 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 0x0E4 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 9-21. Flash Instruction Register

Table 9-24 describes FIR fields.

Table 9-24. 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/two bytes (8b/16b 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/two bytes (8b/16b port) of data from Flash device into next AS field of MDR 1100 CW0—Wait for LFR$\overline{B}$ to return high or time-out, then issue command from FCR[CMD0] 1101 CW1—Wait for LFR$\overline{B}$ to return high or time-out, then issue command from FCR[CMD1] 1110 RBW—Wait for LFR$\overline{B}$ 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$\overline{B}$ to return high or time-out, then read one byte/two bytes (8b/16b 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	

9.3.1.18 Flash Command Register (FCR)

The local bus Flash command register (FCR), shown in Figure 9-22, 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.

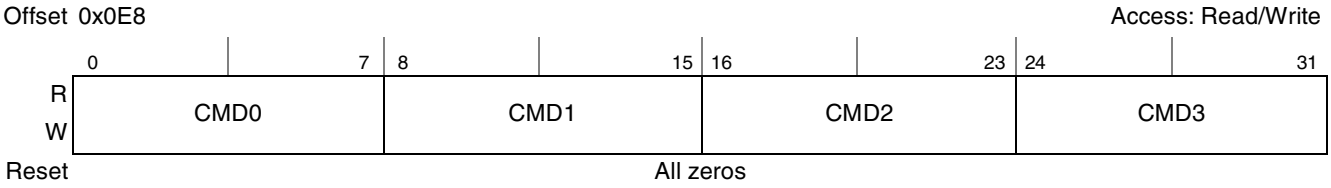


Figure 9-22. Flash Command Register

Table 9-25 describes FCR fields.

Table 9-25. 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.

9.3.1.19 Flash Block Address Register (FBAR)

The local bus Flash block address register (FBAR), shown in Figure 9-23, locates the NAND Flash block index for the page currently accessed.

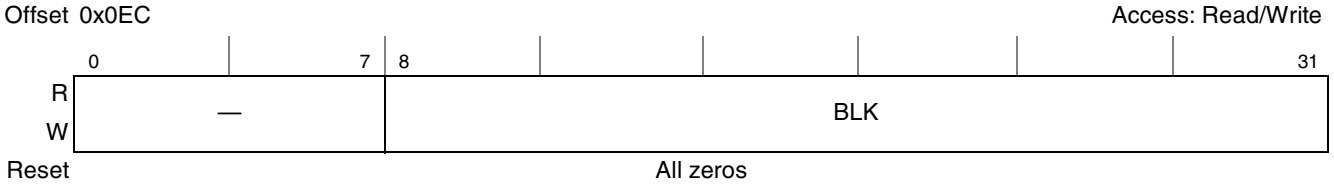


Figure 9-23. Flash Block Address Register

Table 9-26 describes FBAR fields.

Table 9-26. 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.

9.3.1.20 Flash Page Address Register (FPAR)

The local bus Flash page address register (FPAR), shown in Figure 9-24 and Figure 9-25, locates the current NAND Flash page in both the external NAND Flash device and FCM buffer RAM.

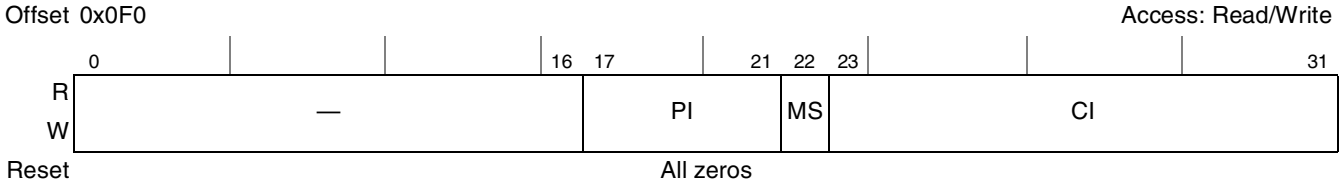


Figure 9-24. Flash Page Address Register, Small Page Device (ORx[PGS] = 0)

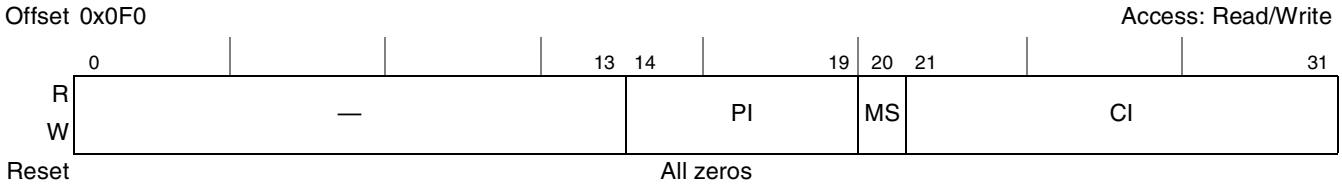


Figure 9-25. Flash Page Address Register, Large Page Device (ORx[PGS] = 1)

Table 9-27 describes FPAR fields for small page devices.

Table 9-27. 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. For a 16-bit port size, the least significant bit of CI is assumed to be zero, hence CI remains a byte index at all times.

Table 9-28 describes FPAR fields for large page devices.

Table 9-28. 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. For a 16-bit port size, the least significant bit of CI is assumed to be zero, hence CI remains a byte index at all times.

9.3.1.21 Flash Byte Count Register (FBCR)

The local bus Flash byte count register (FBCR), shown in Figure 9-26, defines the size of FCM block transfers for reads and writes to the NAND Flash EEPROM.

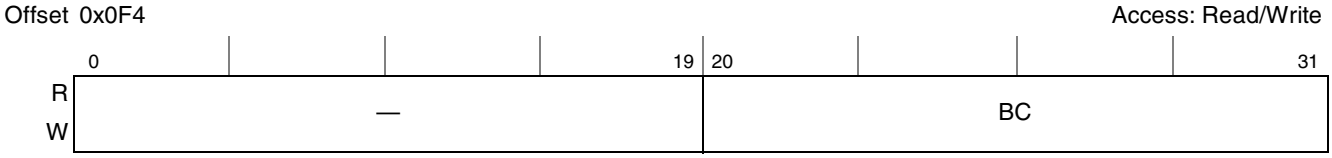


Figure 9-26. Flash Byte Count Register

Table 9-29 describes FBCR fields.

Table 9-29. 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. If the bank port size is 16-bits, the LSB of BC (bit 31) is ignored, thus BC specifies double the number of 16-bit words for transfer. 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.

9.4 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 and 16-bit data buses. 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.

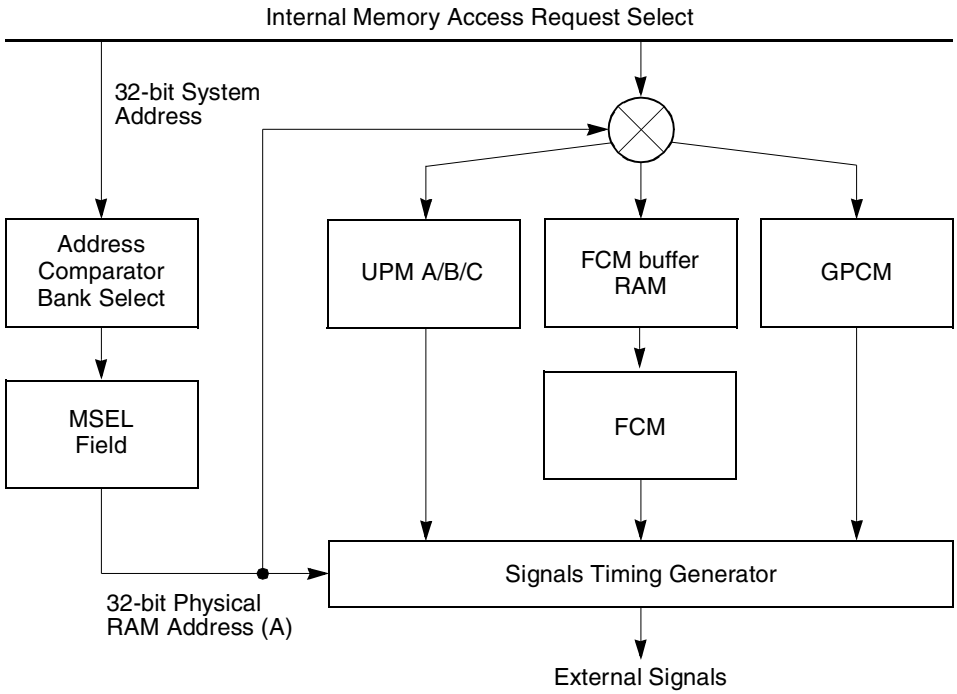


Figure 9-27. Basic Operation of Memory Controllers in the eLBC

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 9-27. 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.

NOTE

Different machines (FCM/GPCM/UPM) share the address, data, and control signals. While the eLBC is servicing a transaction, subsequent transactions are queued until the current transaction has completed.

9.4.1 Basic Architecture

The following subsections describe the basic architecture of the eLBC.

9.4.1.1 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).

9.4.1.2 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. If the eLBC is not used in multiplexed mode, the 26 LSBs of the address are provided by LA[6:31], unlatched, 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 533 MHz, then 1.8 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.

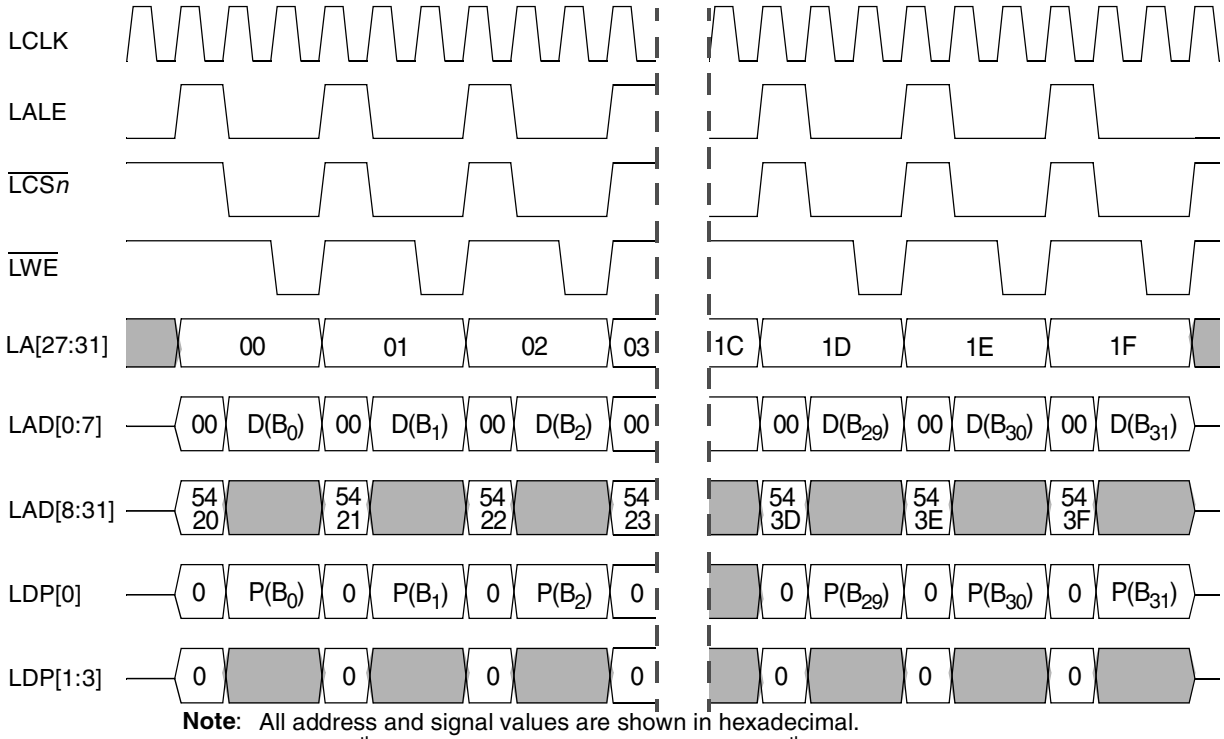
The frequency of LALE assertion varies across the three memory controllers:

- 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 9-28 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.



Note: All address and signal values are shown in hexadecimal.
D(Bk) = kth of 32 data bytes, P(Bk) = parity bit of kth data byte.

Figure 9-28. Example of 8-Bit GPCM Writing 32 Bytes to Address 0x5420 (LCRR[PBYP] = 0)

If the RCW is loaded by the eLBC, LALE may remain at an unknown value for up to 8 cycles after PORESET negation. Thus, LALE should be ignored for 8 CLKIN/PCI_SYNC_IN cycles after PORESET negation. In general, it is recommended that a latch be implemented for this adjustment and not a state machine triggered by LALE.

If the RCW is not loaded by the eLBC (for example, I2C or hard-coded options are used), then LALE is at an unknown value until the PLL is locked and should be ignored until the negation of HRESET.

9.4.1.3 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 MDVAL 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 9-29 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, $\overline{\text{LCS}n}$ (or any other control signal) remains negated or frozen.

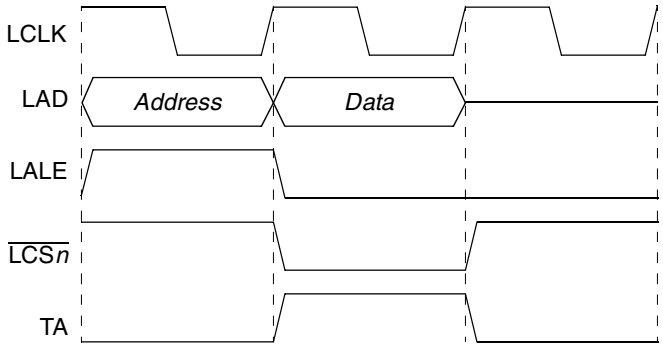


Figure 9-29. Basic eLBC Bus Cycle with LALE, TA, and $\overline{\text{LCS}n}$

9.4.1.4 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 $\overline{\text{LWE}}$ or an extra $\overline{\text{LOE}}$ 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.

9.4.1.5 Atomic Operation

The eLBC supports the following kinds of atomic bus operations (set by BR*n*[ATOM]):

- Read-after-write atomic (RAWA). When a write access hits a memory bank in which ATOM = 01, the eLBC reserves the selected memory bank for the exclusive use of the accessing master.

  While the bank is reserved, no other device can be granted access to this bank. The reservation is released when the master that created it accesses the same bank with a read transaction. Additional write transactions prior to the releasing read do not change reservation status, but are otherwise processed normally. If the master fails to release the reservation within 256 bus clock cycles, the reservation is released and an atomic error is reported (if enabled); additional write transactions prior to the releasing read restart the reservation timer. This feature is intended for CAM operations.
- Write-after-read atomic (WARA). When a read access hits a memory bank in which ATOM = 10, the eLBC reserves the bus for the exclusive use of the accessing master.

During the reservation period, no other device can be granted access to the atomic bank. The reservation is released when the device that created it accesses the same bank with a write transaction. Additional read transactions prior to the releasing write are otherwise processed normally and do not change the reservation status. If the device fails to release the reservation within 256 bus clock cycles, the reservation is released and an atomic error is reported (if enabled); additional read transactions prior to the releasing write restart the reservation timer.

9.4.1.6 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).

9.4.1.7 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 (that is, 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 9.4.4.1.4, “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.

NOTE

When the FCM is in the middle of a long transaction (such as NAND erase, write, etc.), another transaction on the GPCM or UPM will trigger the bus monitor to start, even though the GPCM or UPM is waiting for the FCM to finish. If the bus monitor times out, it could corrupt the current NAND flash operation as well as terminate the GPCM or UPM operation. To avoid such cases, it is recommended that the bus monitor timeout be programmed to its maximum setting of LBCR[BMT] = 0 and LBCR[BMTPS] = 0xF.

9.4.2 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 9-30 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.

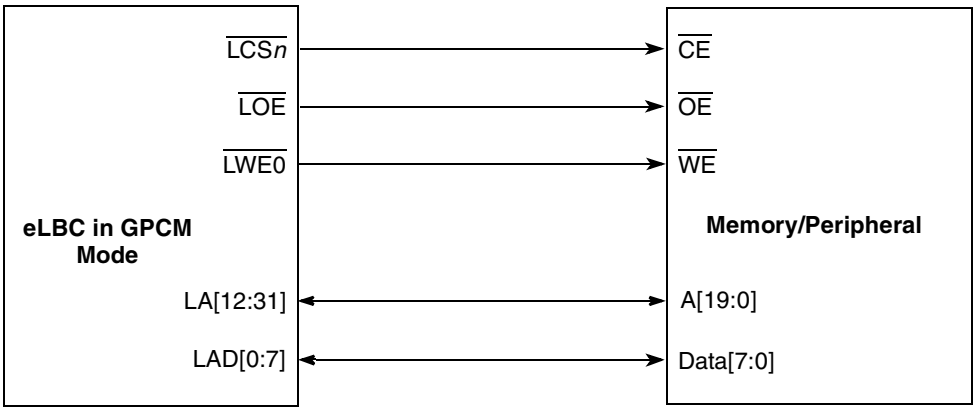


Figure 9-30. Enhanced Local Bus to GPCM Device Interface

Figure 9-31 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.

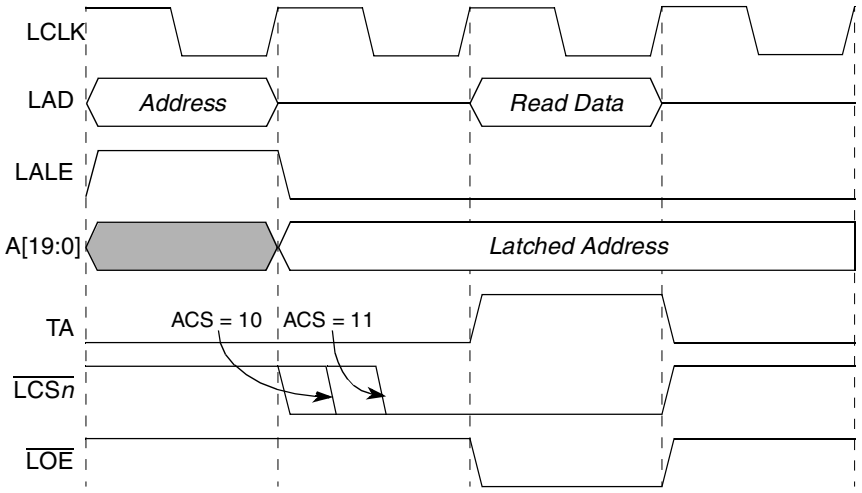
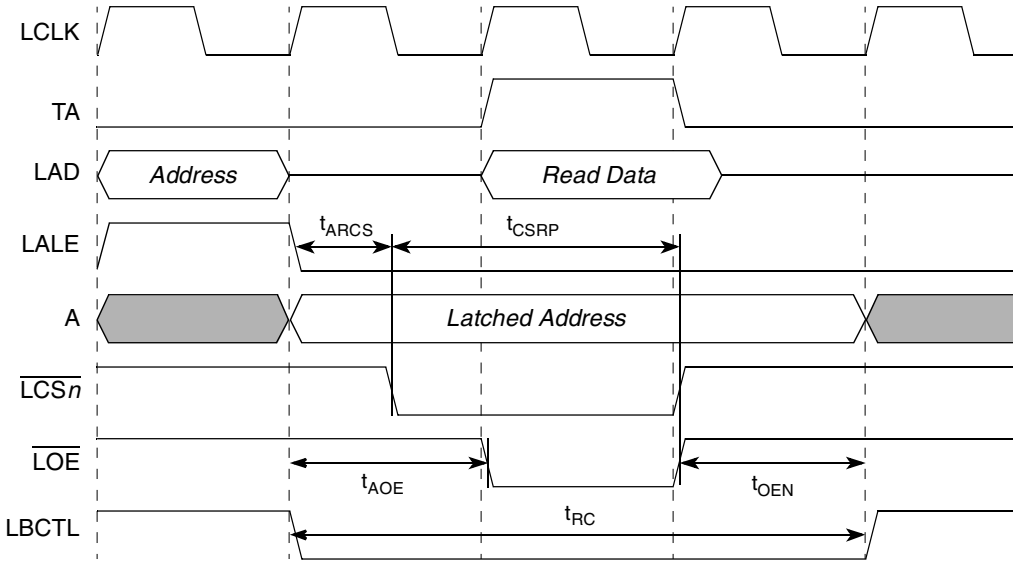


Figure 9-31. GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0)

9.4.2.1 GPCM Read Signal Timing

The basic GPCM read timing parameters that may be set by the OR*n* attributes are shown in Figure 9-32. 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 9-32. GPCM General Read Timing Parameters

Table 9-30 lists the signal timing parameters for a GPCM read access as the option register attributes are varied.

Table 9-30. GPCM Read Control Signal Timing

Option Register Attributes				Signal Timing (LCLK clock cycles)				
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	1¾ + 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	1¾ + SCY	1	1	3 + SCY

Table 9-30. GPCM Read Control Signal Timing (continued)

Option Register Attributes				Signal Timing (LCLK clock cycles)				
TRLX	EHTR	XACS	ACS	t_{ARCS}	t_{CSRP}	t_{AOE}	t_{OEN}	t_{RC}
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	1¾ + 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	1¾ + 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

9.4.2.2 GPCM Write Signal Timing

The basic GPCM write timing parameters that may be set by the OR*n* attributes are shown in Figure 9-33. The write access cycle commences upon latching of the memory address (LALE negated), and concludes when $\overline{\text{LCS}n}$ 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. $\overline{\text{LWE}}$ may, in some cases, negate high before the end of the write access to provide additional hold time for the external memory.

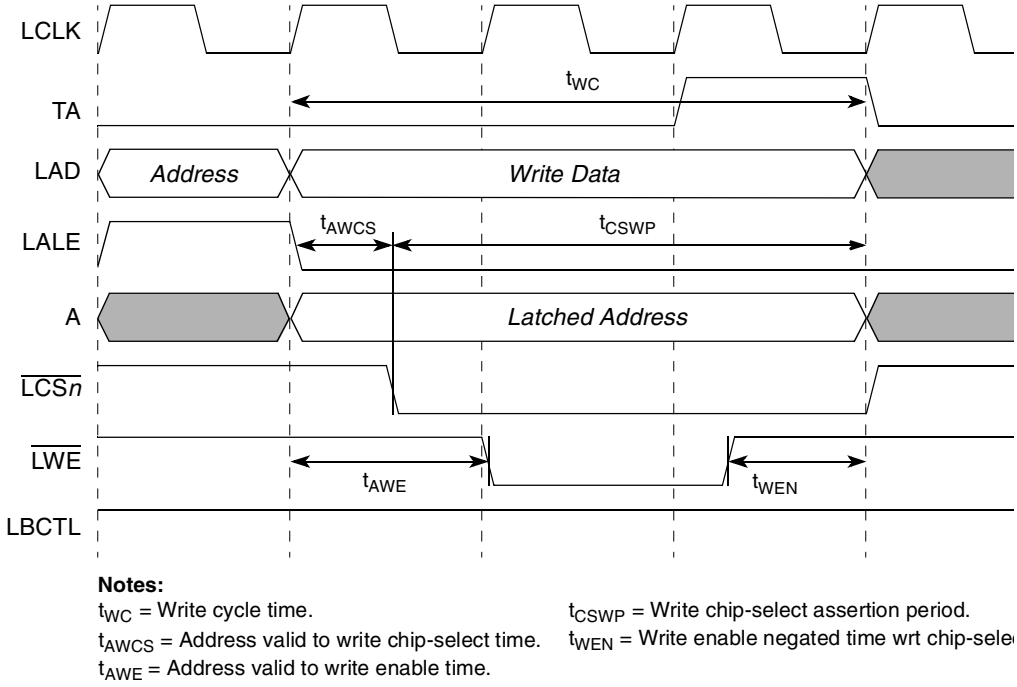


Figure 9-33. GPCM General Write Timing Parameters

Table 9-31 lists the signal timing parameters for a GPCM write access as the option register attributes are varied.

Table 9-31. GPCM Write Control Signal Timing

Option Register Attributes				Signal Timing (LCLK clock cycles)				
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	1¾ + 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	2 + SCY
0	0	10	1	¼	1½ + SCY	1	0	1¾ + SCY

Table 9-31. GPCM Write Control Signal Timing (continued)

Option Register Attributes				Signal Timing (LCLK clock cycles)				
TRLX	XACS	ACS	CSNT	t_{AWCS}	t_{CSWP}	t_{AWE}	t_{WEN}	t_{WC}
0	0	11	1	½	1¼ + SCY	1	0	1¾ + SCY
0	1	00	1	0	2 + SCY	1	¼	2 + SCY
0	1	10	1	1	¾ + SCY	1	0	1¾ + SCY
0	1	11	1	2	¾ + SCY	2	0	2¾ + SCY
1	0	00	0	0	2 + 2 × SCY	1	0	2 + 2 × SCY
1	0	10	0	1¼	1¾ + 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	3 + 2 × SCY
1	0	10	1	1¼	1½ + 2 × SCY	2	0	2¾ + 2 × SCY
1	0	11	1	1½	1¼ + 2 × SCY	2	0	2¾ + 2 × SCY
1	1	00	1	0	3 + 2 × SCY	1	1¼	3 + 2 × SCY
1	1	10	1	2	¾ + 2 × SCY	2	0	2¾ + 2 × SCY
1	1	11	1	3	¾ + 2 × SCY	3	0	3¾ + 2 × SCY

9.4.2.3 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.
- One half of a clock cycle later.
- One clock cycle later (for LCRR[CLKDIV] = 2 (clock ratio of 4)), when OR*n*[XACS] = 1.
- Two clock cycles later, when OR*n*[XACS] = 1.
- Three clock cycles later, when OR*n*[XACS] = 1 and OR*n*[TRLX] = 1.

The timing diagram in Figure 9-31 shows two chip-select assertion timings.

9.4.2.3.1 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.

9.4.2.3.2 Chip-Select and Write Enable Negation Timing

Figure 9-30 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.

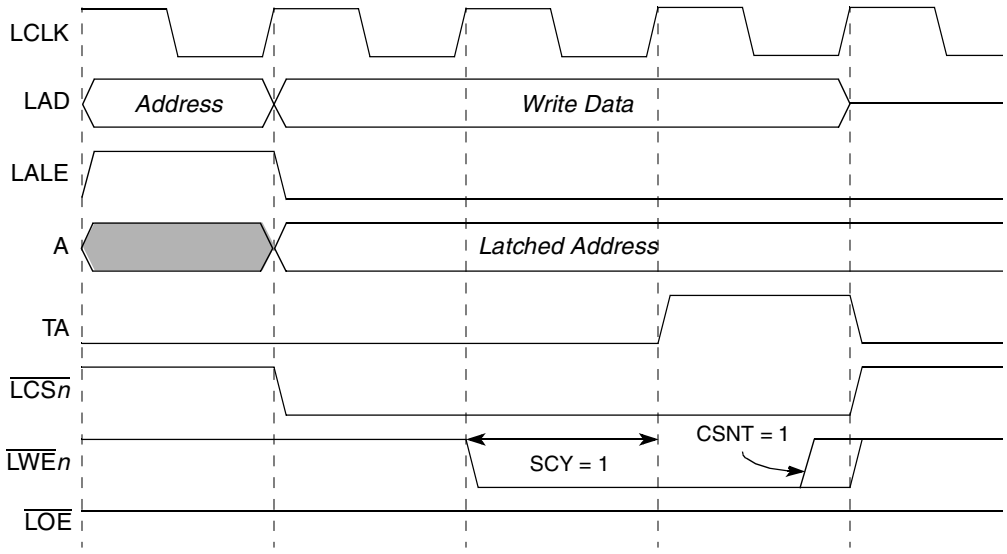


Figure 9-34. GPCM Basic Write Timing
(XACS = 0, ACS = 00, CSNT = 1, SCY = 1, TRLX = 0)

As Figure 9-34 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. For example, when ACS = 00 and CSNT = 1, $\overline{\text{LWE}}$*n* is negated one quarter of a clock earlier, as shown in Figure 9-34.

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.
3. TRLX = 1 in conjunction with CSNT = 1, negates the $\overline{\text{LCS}}$*n* and $\overline{\text{LWE}}$*n* 1 + 1/4 cycle earlier.

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 9-34.

9.4.2.3.3 Relaxed Timing

ORx[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 9-35 and Figure 9-36 show relaxed timing read and write transactions. The example in Figure 9-36 also shows address and data multiplexing on LAD for a pair of writes issued consecutively.

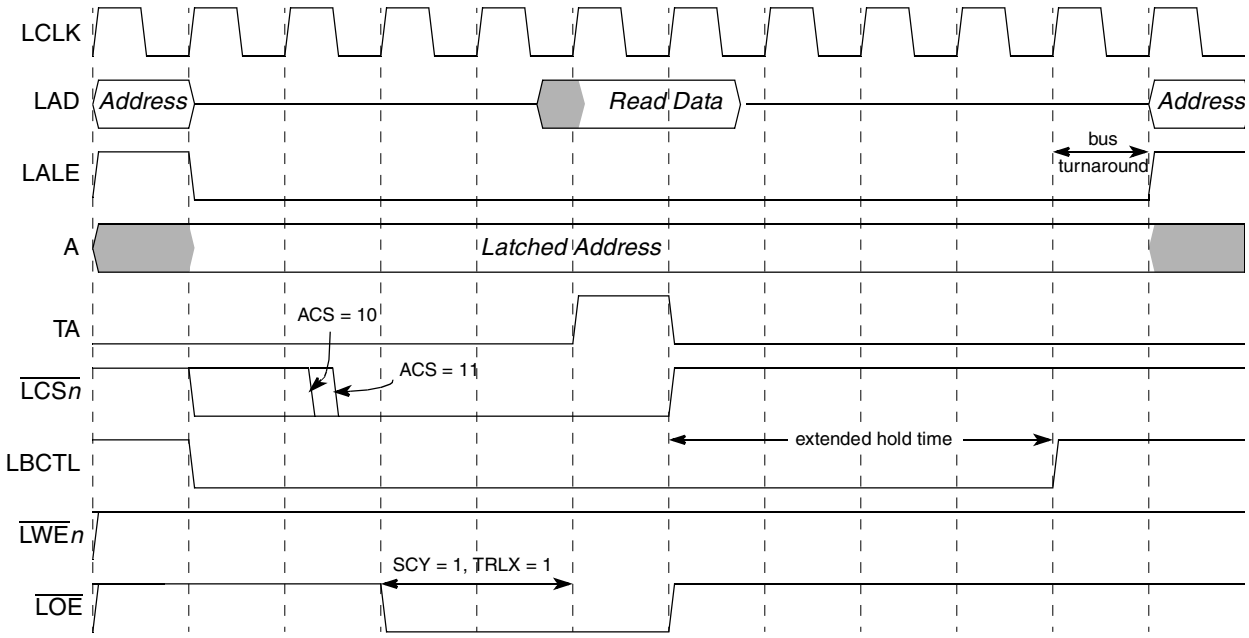


Figure 9-35. GPCM Relaxed Timing Back-to-Back Reads (XACS = 0, ACS = 1x, SCY = 1, CSNT = 0, TRLX = 1, EHTR = 0)

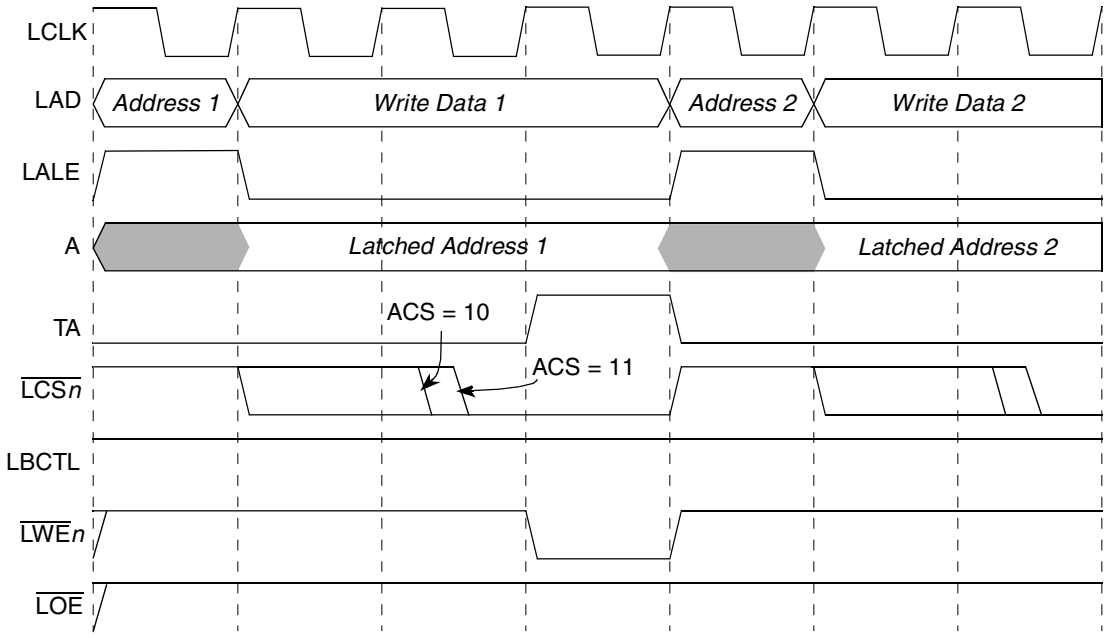


Figure 9-36. GPCM Relaxed Timing Back-to-Back Writes (XACS = 0, ACS = 1x, SCY = 0, CSNT = 0, TRLX = 1)

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 9-37 and Figure 9-38. If ACS ≠ 00, $\overline{\text{LCS}}n$ is also negated one clock earlier.

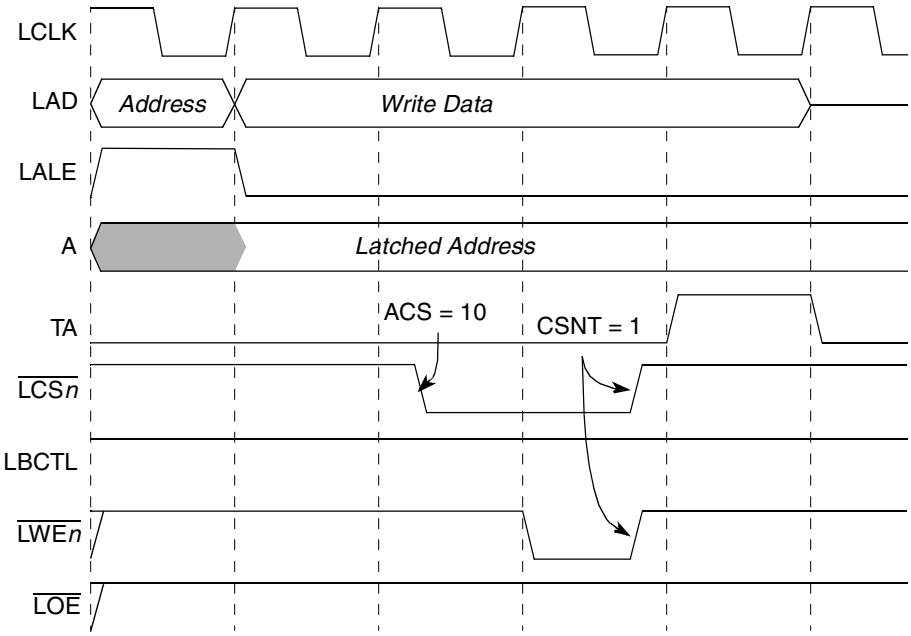


Figure 9-37. GPCM Relaxed Timing Write (XACS = 0, ACS = 10, SCY = 0, CSNT = 1, TRLX = 1)

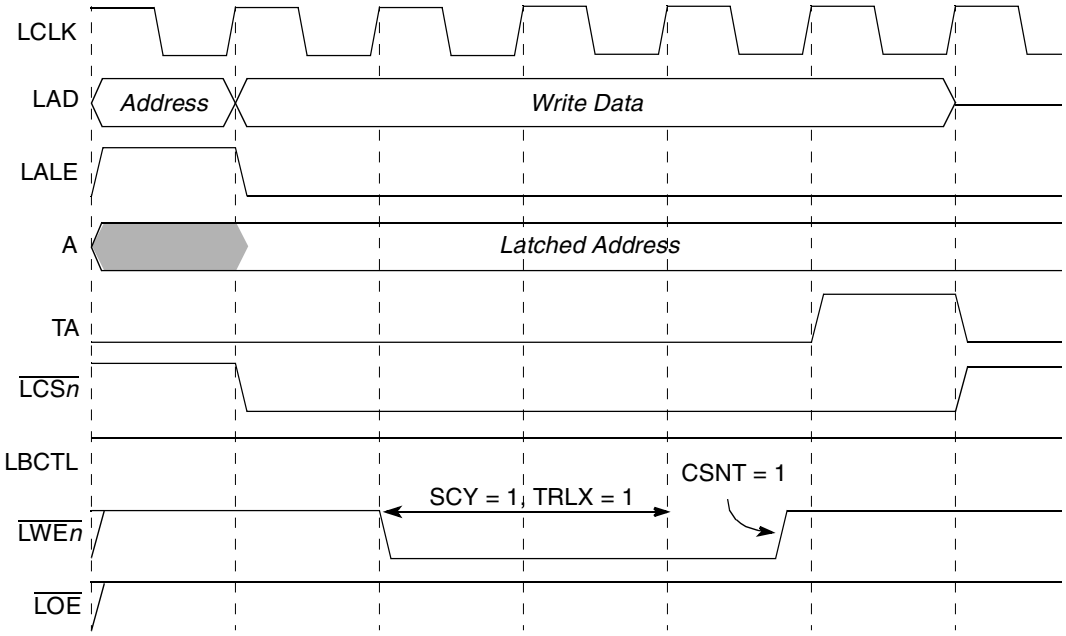


Figure 9-38. GPCM Relaxed Timing Write
(XACS = 0, ACS = 00, SCY = 1, CSNT = 1, TRLX = 1)

9.4.2.3.4 Output Enable ($\overline{\text{LOE}}$) Timing

The timing of the $\overline{\text{LOE}}$ is affected only by TRLX. It always asserts and negates on the rising edge of the bus clock. $\overline{\text{LOE}}$ asserts either on the rising edge of the bus clock after $\overline{\text{LCS}n}$ is asserted or coinciding with $\overline{\text{LCS}n}$ (if XACS = 1 and ACS = 10 or ACS = 11). Accordingly, assertion of $\overline{\text{LOE}}$ can be delayed (along with the assertion of $\overline{\text{LCS}n}$) by programming TRLX = 1. $\overline{\text{LOE}}$ negates on the rising clock edge coinciding with $\overline{\text{LCS}n}$ negation

9.4.2.3.5 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 9-7 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].

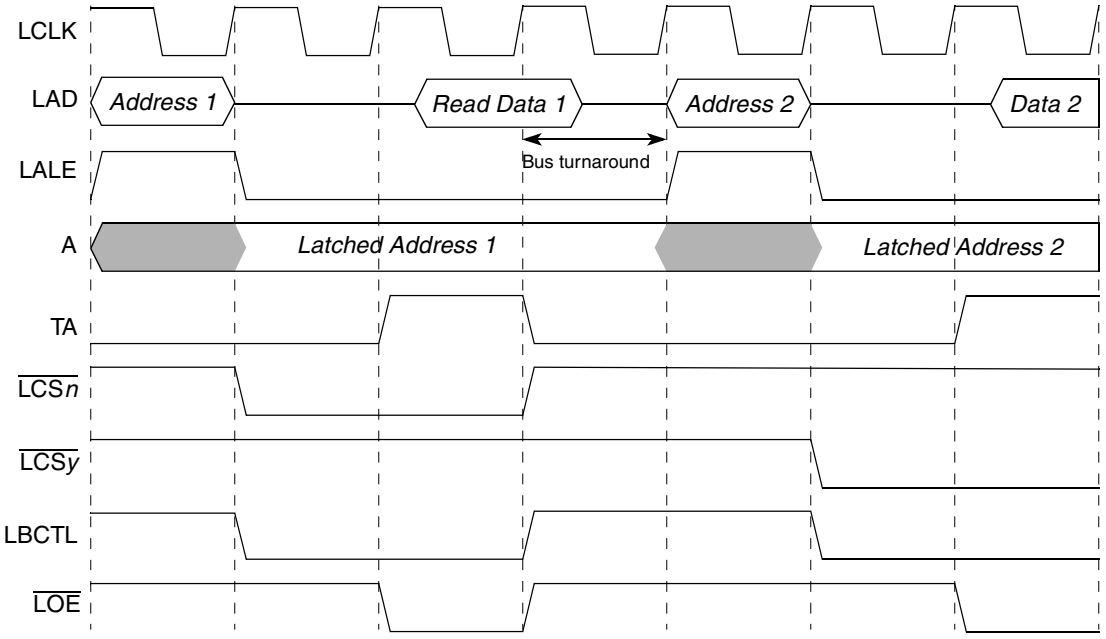


Figure 9-39. GPCM Read Followed by Read (TRLX = 0, EHTR = 0, Fastest Timing)

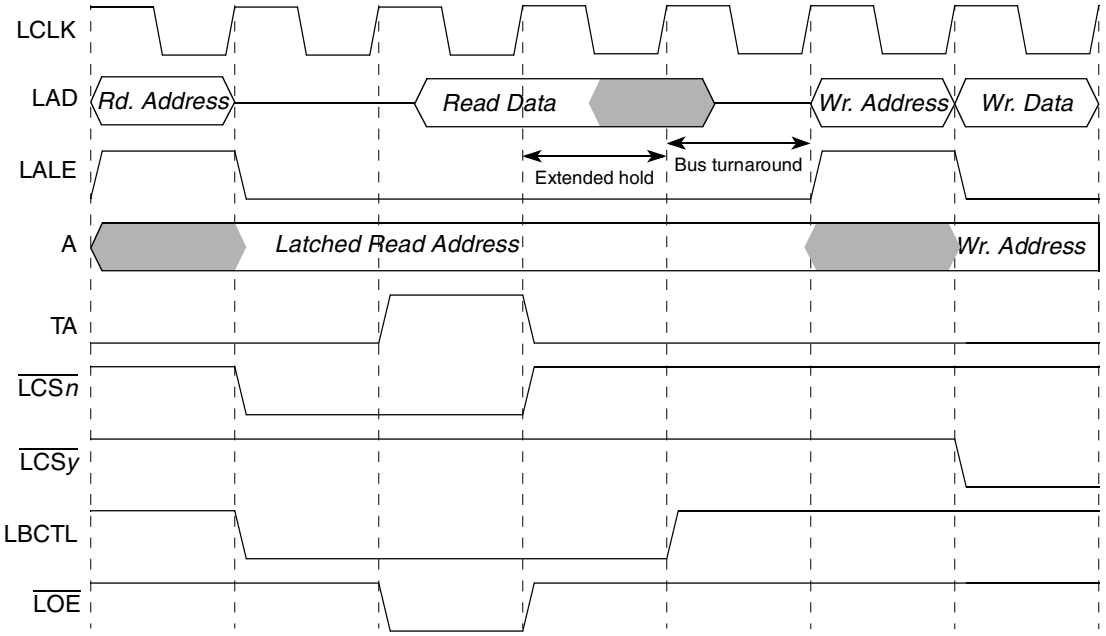


Figure 9-40. GPCM Read Followed by Write
(TRLX = 0, EHTR = 1, One-Cycle Extended Hold Time on Reads)

9.4.2.4 External Access Termination ($\overline{\text{LGTA}}$)

External access termination is supported by the GPCM using the asynchronous $\overline{\text{LGTA}}$ input signal, which is synchronized and sampled internally by the local bus. If, during assertion of $\overline{\text{LCS}n}$, the sampled $\overline{\text{LGTA}}$ 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]). $\overline{\text{LGTA}}$ should be asserted for at least one bus cycle to be effective. Note that because $\overline{\text{LGTA}}$ is synchronized, bus termination occurs two cycles after $\overline{\text{LGTA}}$ assertion, so in case of read cycle, the device still must drive data as long as $\overline{\text{LOE}}$ is asserted.

The user selects whether transfer acknowledge is generated internally or externally ($\overline{\text{LGTA}}$) by programming OR*n*[SETA]. Asserting $\overline{\text{LGTA}}$ 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.

In PLL bypass mode, the timing of $\overline{\text{LGTA}}$ is illustrated by the example in Figure 9-41.

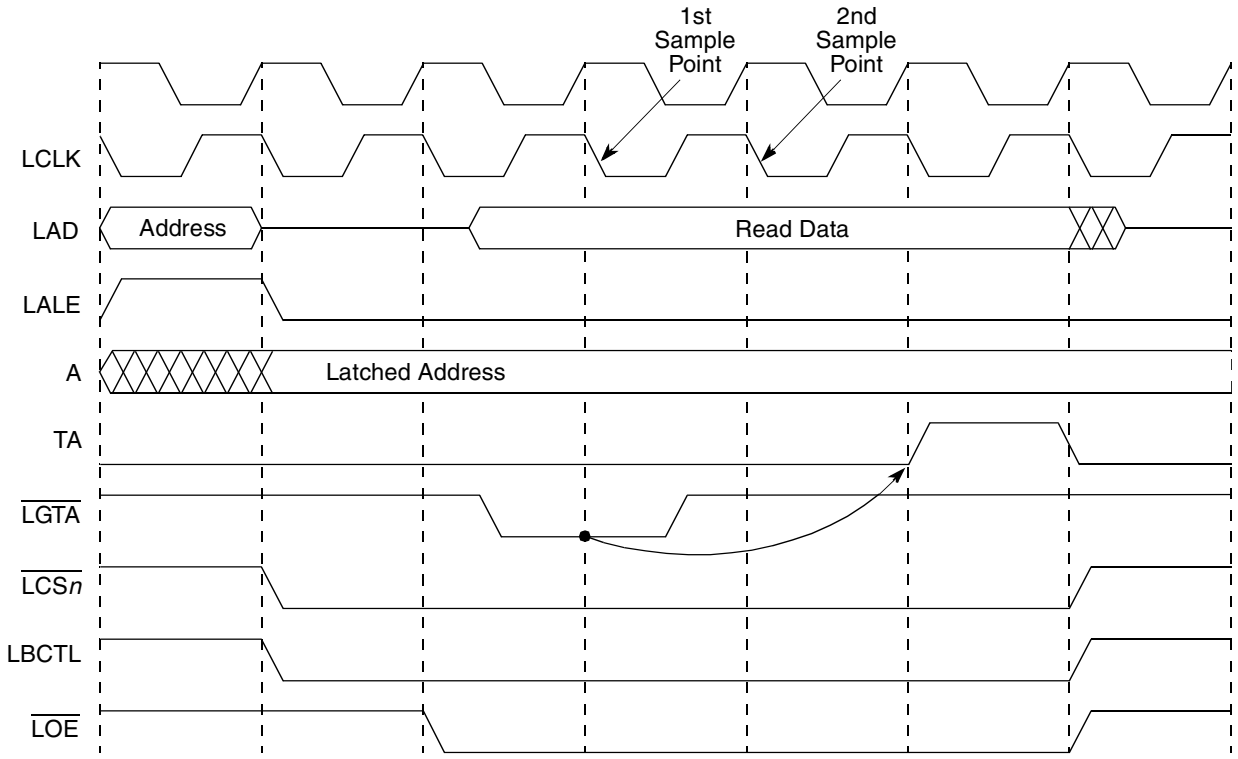


Figure 9-41. External Termination of GPCM Access (PLL Bypass Mode)

9.4.2.5 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 9-32 describes the initial values of the boot bank in the memory controller.

Table 9-32. Boot Bank Field Values after Reset for GPCM as Boot Controller

Register	Field	Setting
BR0	BA	0000_0000_0000_0000_0
	PS	*From cfg_rom_loc*
	DECC	00
	WP	0
	MSEL	000
	ATOM	00
	V	1
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

9.4.3 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 9-42 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 ($\overline{\text{LCS0}}$) prior to the system being fully configured.

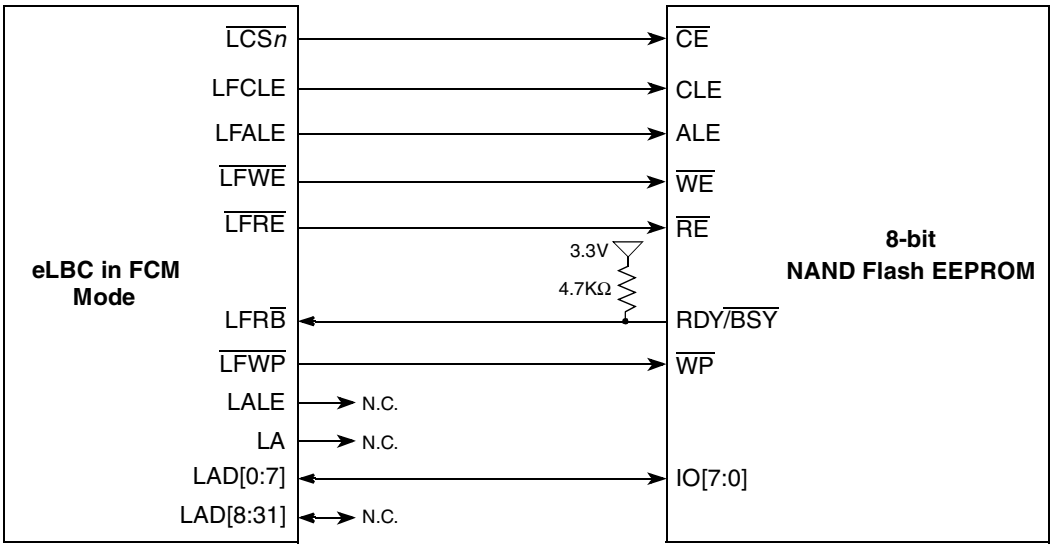


Figure 9-42. Local Bus to 8-Bit FCM Device Interface

Figure 9-43 shows connection of a 16-bit NAND Flash EEPROM to eLBC in FCM mode. Commands, address bytes, and the high byte of data appear on LAD[0:7], as in the 8-bit case, whereas the low byte of 16-bit data is placed on LAD[8:15] during read and write data transfers.

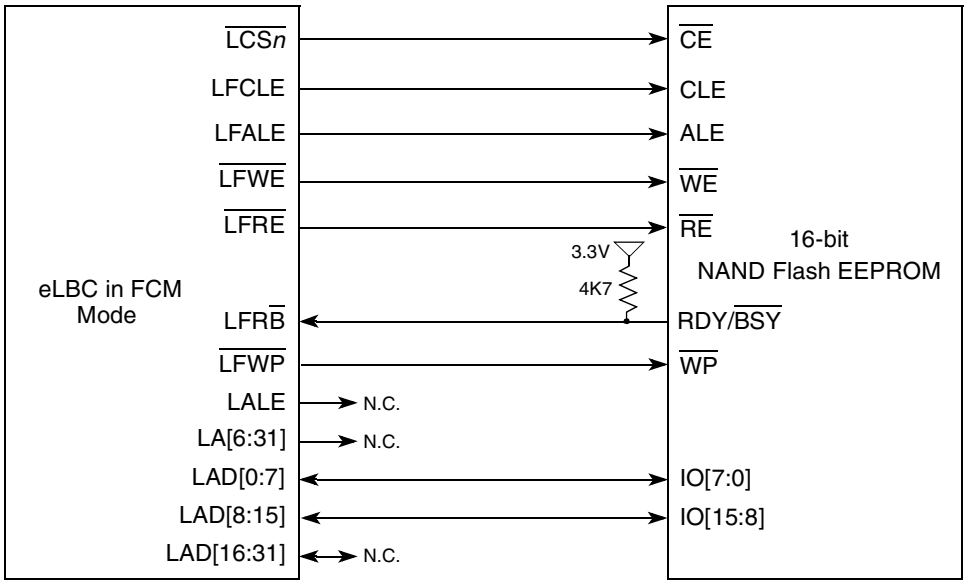


Figure 9-43. Local Bus to 16-bit FCM Device Interface

Basic read access timing for FCM is shown in Figure 9-44. Although LCLK is shown for reference, NAND Flash EEPROMs do not make use of the clock.

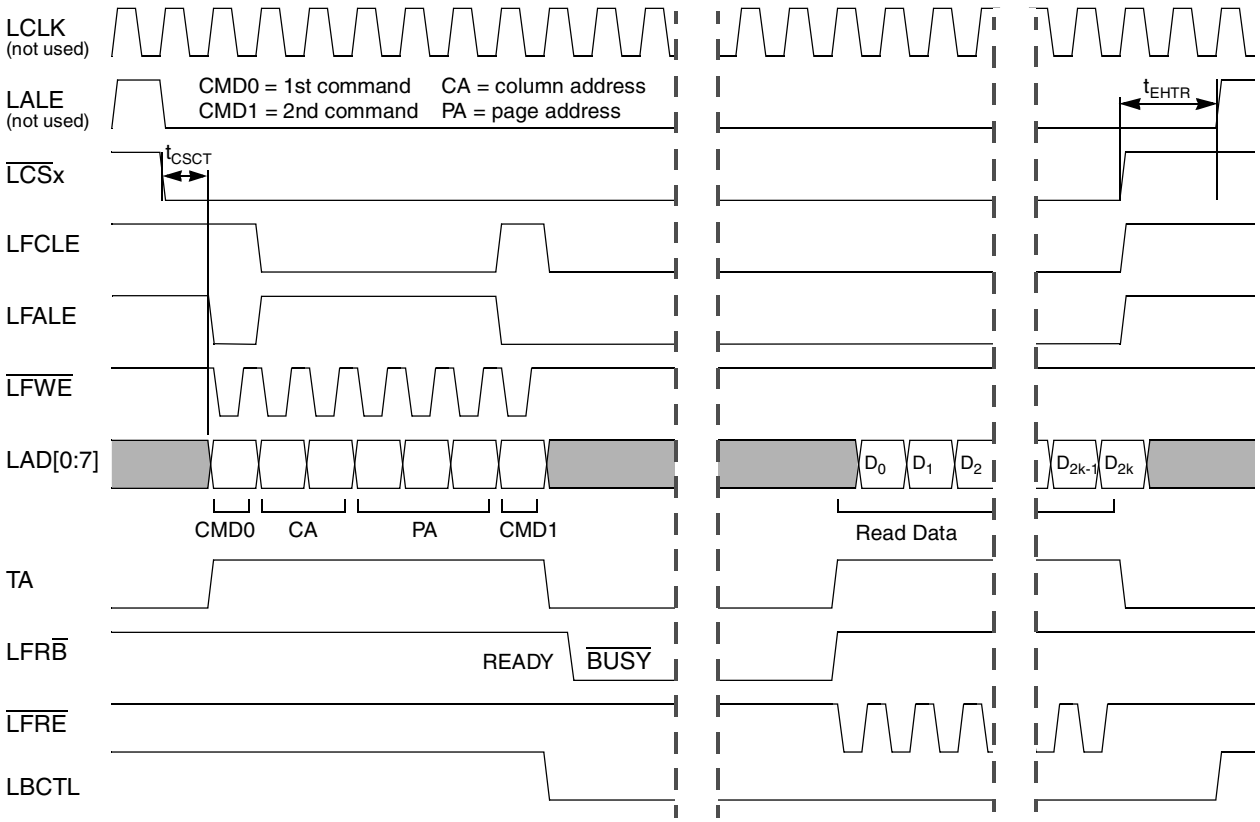


Figure 9-44. FCM Basic Page Read Timing
(PGS = 1, CSCT = 0, CST = 0, CHT = 1, RST = 1, SCY = 0, TRLX = 0, EHTR = 1)

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 $\text{LFR}\overline{\text{B}}$ low to indicate that the device is busy. Since in Figure 9-44 FCM is now expecting a read response, it takes LBCTL low to turnaround any bus transceivers that are present. Upon $\text{LFR}\overline{\text{B}}$ 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.

9.4.3.1 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 9.4.3.4.2, “Boot Block Loading into the FCM Buffer RAM,” for a description of the shared buffer RAM layout during boot.

9.4.3.1.1 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 9-45. 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.

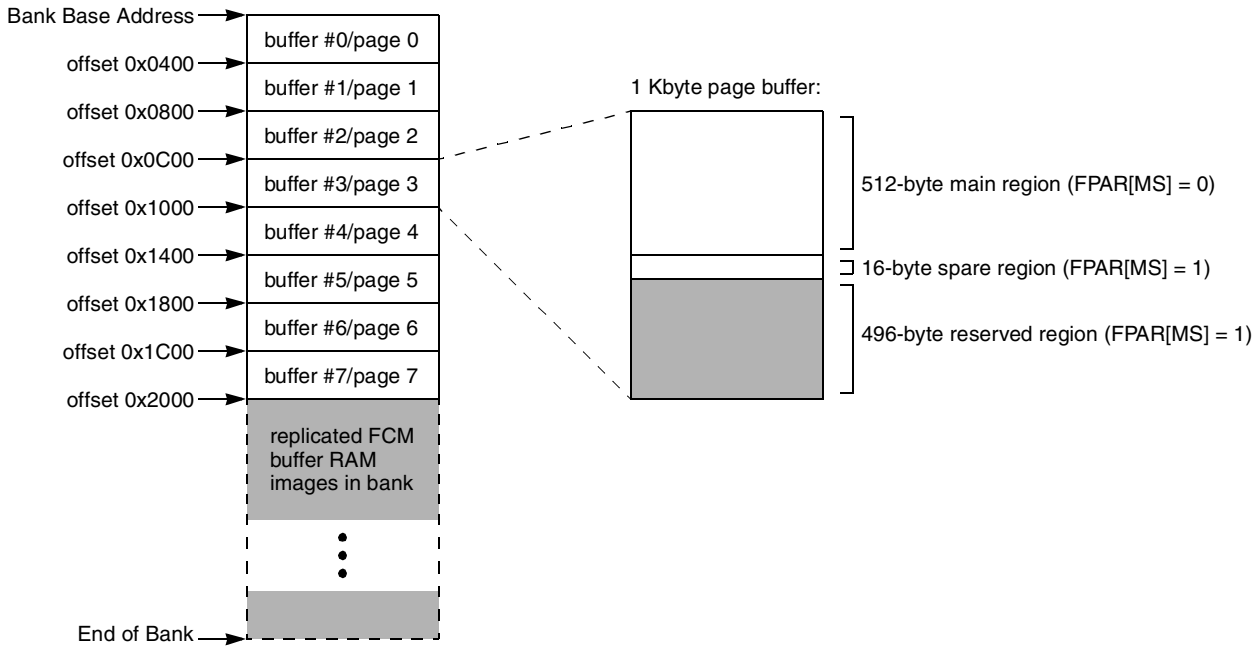


Figure 9-45. FCM Buffer RAM Memory Map for Small-Page (512-Byte page) NAND Flash Devices

9.4.3.1.2 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 9-46. 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.

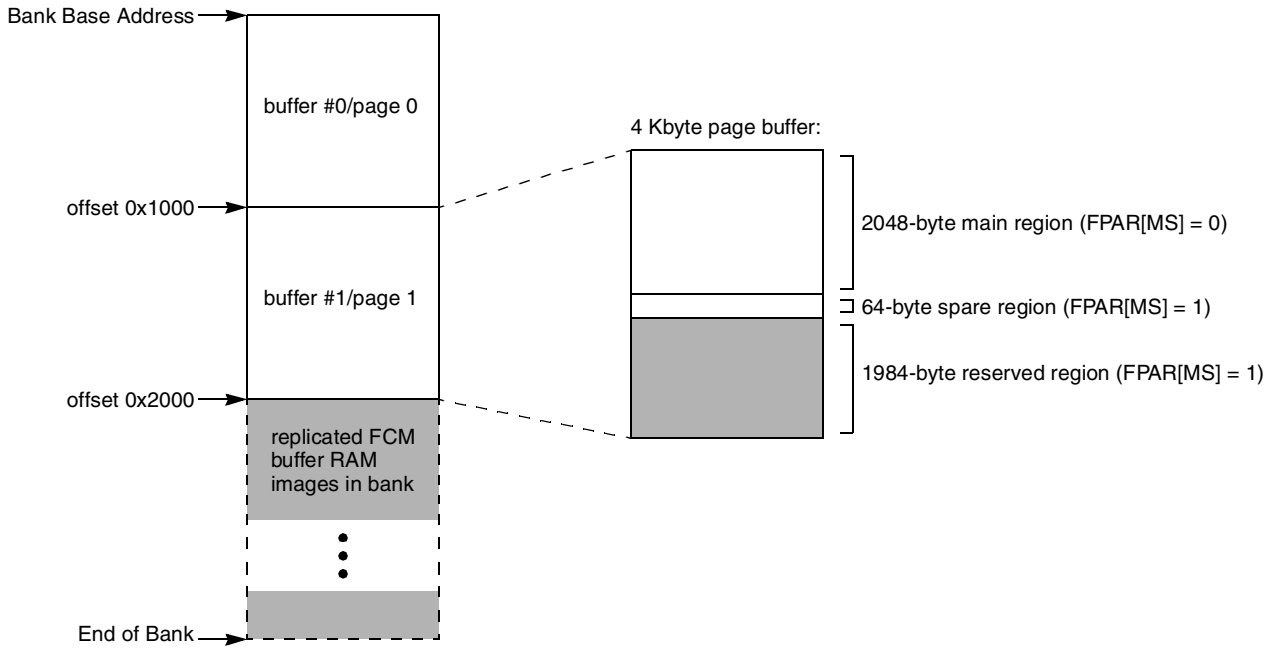


Figure 9-46. FCM Buffer RAM Memory Map for Large-Page (2-Kbyte page) NAND Flash Devices

9.4.3.1.3 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 9-47. 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.).

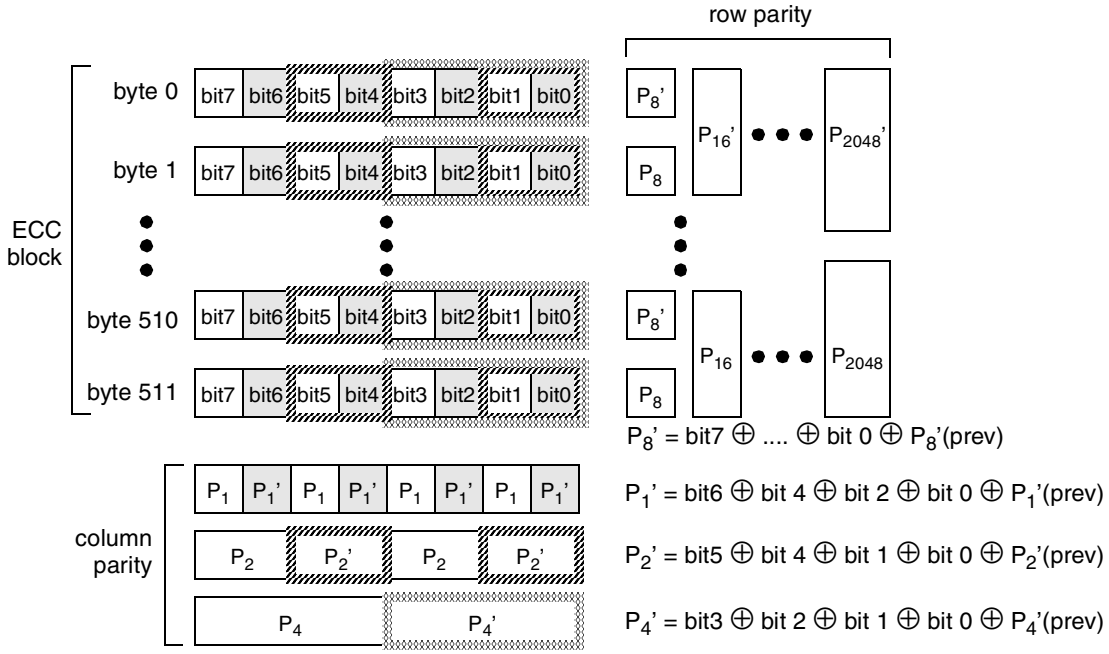


Figure 9-47. FCM ECC Calculation

The 24-bit ECC code word format is shown in Figure 9-48 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	~P_{64}	~P_{64}'	~P_{32}	~P_{32}'	~P_{16}	~P_{16}'	~P_8	~P_8'
EC1	~P_{1024}	~P_{1024}'	~P_{512}	~P_{512}'	~P_{256}	~P_{256}'	~P_{128}	~P_{128}'
EC2	~P_4	~P_4'	~P_2	~P_2'	~P_1	~P_1'	~P_{2048}	~P_{2048}'

Figure 9-48. ECC Layout for LBCR[EPAR] = 0 (~ Represents Logical Negation)

The placement of ECC code words in relation to FMR[ECCM] is shown in Figure 9-49. 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 9-49. 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 non-correctable.

ECCM	Byte 0	Byte 511	Other Mains	Spare 0	5	6	7	8	9	10	11	12	13	14	15
0	Main Region			—		EC0	EC1	EC2	—						
1	Main Region			—				EC0	EC1	EC2	—				

Figure 9-49. ECC Placement in NAND Flash Spare Regions in Relation to FMR[ECCM]

9.4.3.2 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 9-50. 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 $\overline{\text{LCS}n}$, otherwise, $\overline{\text{LCS}n}$ 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 9.3.1.17, "Flash Instruction Register (FIR)."

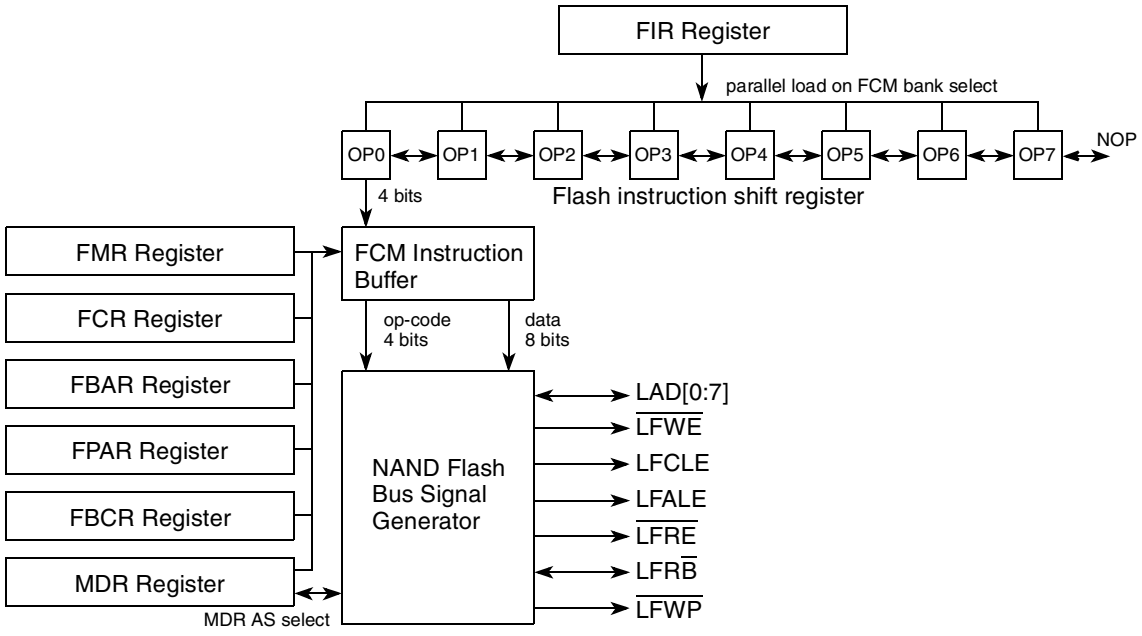


Figure 9-50. FCM Instruction Sequencer Mechanism

9.4.3.2.1 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.

9.4.3.2.2 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.

9.4.3.2.3 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.

9.4.3.2.4 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.Note that for transfers from 16-bit devices FBCR[BC] continues to specify the byte count, but one that is assumed to be a multiple of two. 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 either one byte (8-bit port size) or two bytes (16-bit port size) of data into the next one or two AS fields of MDR.For 16-bit devices, the data on LAD[0:7] is read into MDR[AS_n], while the data on LAD[8:15] is read into MDR[AS_{n+1}], where *n* equals the current MDR read pointer. 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.

9.4.3.2.5 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].Note that for transfers to 16-bit devices FBCR[BC] continues to specify the byte count, but one that is assumed to be a multiple of two. 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 either one byte (8-bit port size) or two bytes (16-bit port size) of data taken from the next one or two AS fields of MDR.For 16-bit devices, the data on LAD[0:7] is taken from MDR[AS_n], while the data on LAD[8:15] is taken from MDR[AS_{n+1}], where *n* equals the current MDR write pointer. 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.

9.4.3.3 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.

9.4.3.3.1 FCM Chip-Select Timing

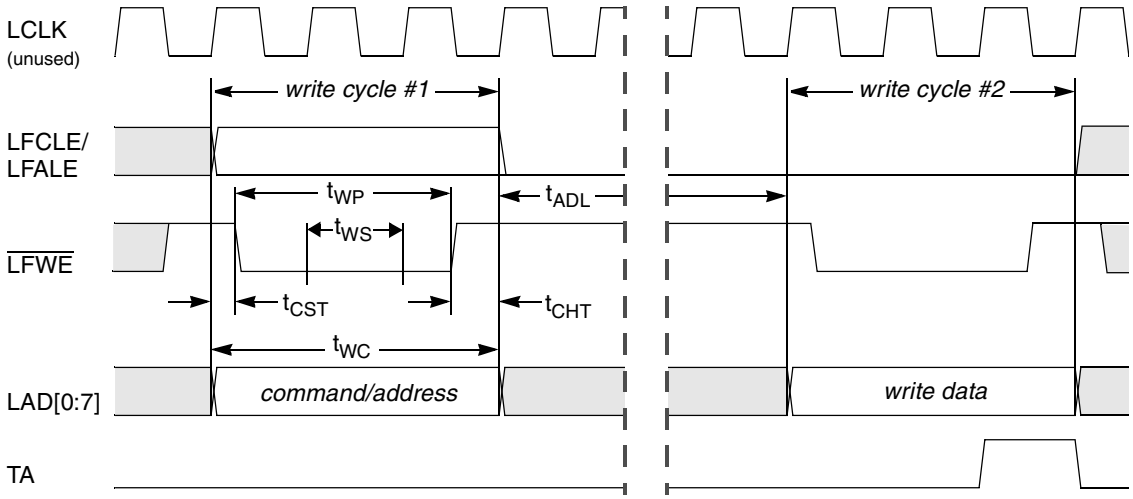
The timing of $\overline{\text{LCS}n}$ assertion in FCM mode is illustrated by the timing diagram in Figure 9-44. $\overline{\text{LCS}n}$ is asserted immediately following LALE negation, and remains asserted until the last instruction in FIR has completed. The delay, t_{CSCT}, between $\overline{\text{LCS}n}$ assertion and commencement of the first NAND Flash instruction is controlled by OR*n*[CSCT] and OR*n*[TRLX], as shown in Table 9-33. OR*n*[CSCT] should be set in accordance with the NAND Flash EEPROM chip-select to $\overline{\text{WE}}$ set-up time specification.

Table 9-33. FCM Chip-Select to First Command Timing

ORn[TRLX]	ORn[CSCT]	$\overline{\text{LCS}n}$ 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

9.4.3.3.2 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 $\overline{\text{LFWE}}$ 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 $\overline{\text{LFWE}}$, as shown in Figure 9-51.



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 9-51. Timing of FCM Command/Address and Write Data Cycles (for TRLX = 0, CHT = 0, CST = 1, SCY = 1)

The timing parameters are summarized in Table 9-34.

Table 9-34. 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 9-52. 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.

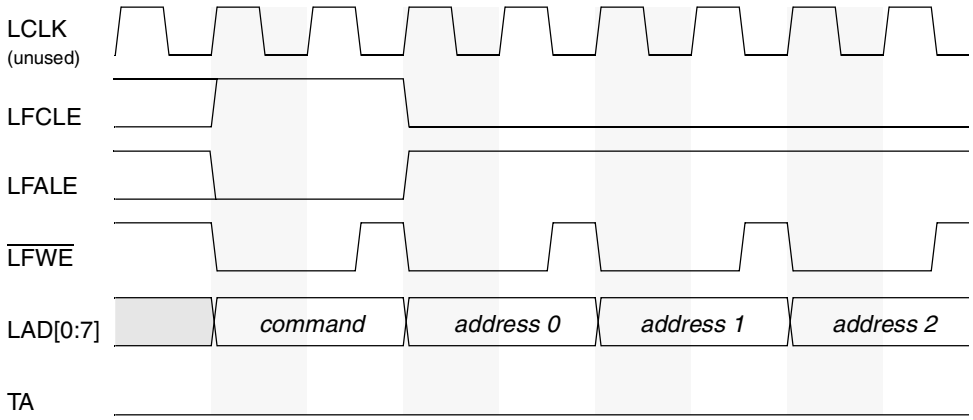


Figure 9-52. Example of FCM Command and Address Timing with Minimum Delay Parameters (for TRLX = 0, CHT = 0, CST = 0, SCY = 0)

An example of relaxed command timing is shown in Figure 9-53.

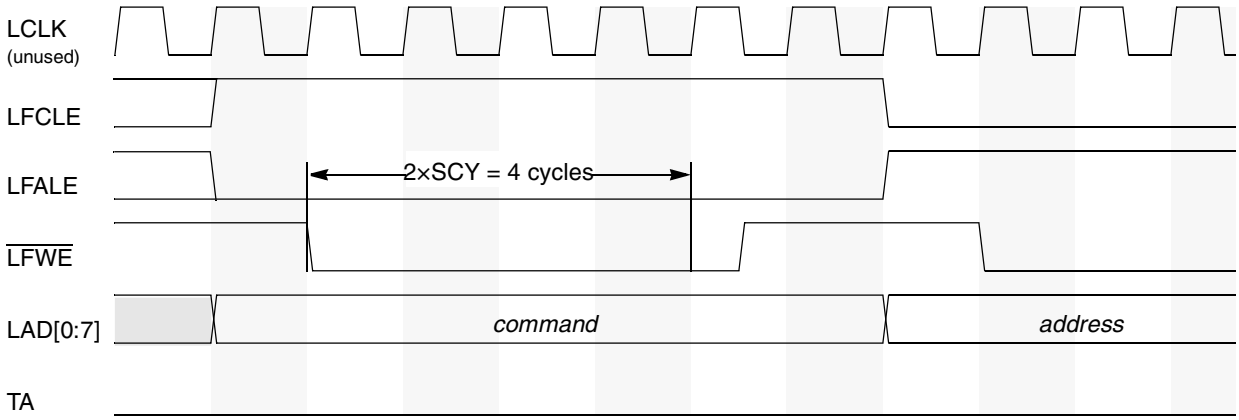


Figure 9-53. Example of FCM Command and Address Timing with Relaxed Parameters (for TRLX = 1, CHT = 0, CST = 1, SCY = 2)

9.4.3.3.3 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 9-54 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 LFRB on two consecutive cycles of LCLK to filter out noise on this signal as it rises to the ready state (LFRB = 1).

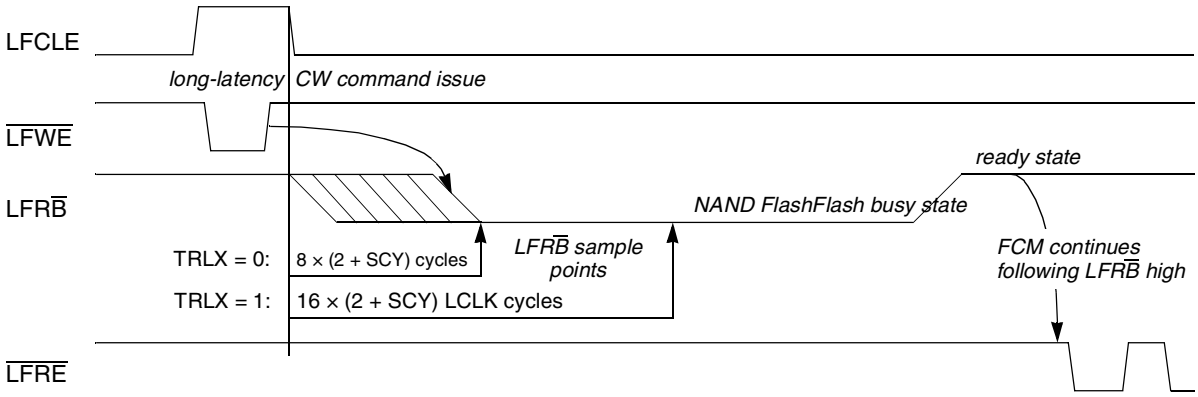
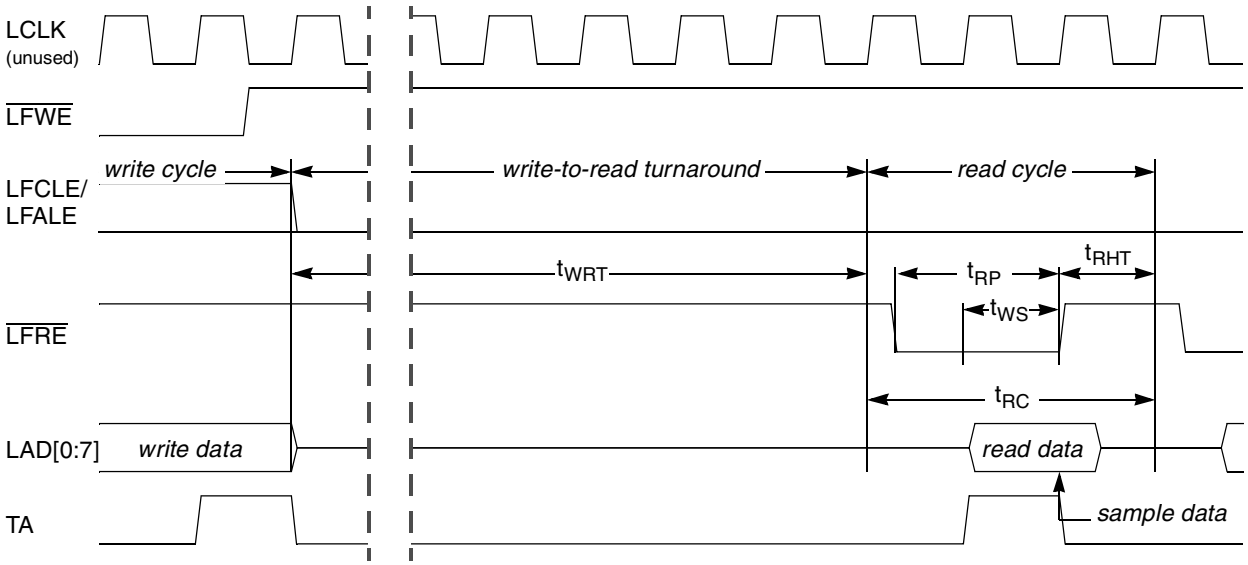


Figure 9-54. FCM Delay Prior to Sampling LFRB State

9.4.3.3.4 FCM Read Data Timing

The timing for read data transfers is shown in Figure 9-55. Upon assertion of LFRE, the Flash device will enable its output drivers and drive valid read data while LFRE is held low. FCM samples read data on the rising edge of 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 LFRB sample timing takes effect (see Section 9.4.3.3.3, "FCM Ready/Busy Timing").



Notes: t_{RP} = LFRE pulse time, read period. t_{WS} = Read wait state time.
t_{RHT} = LFRE hold time. t_{RC} = Read data cycle time.
t_{WRT} = Write to read turnaround time.

Figure 9-55. FCM Read Data Timing (for TRLX = 0, RST = 0, SCY = 1)

The timing parameters are summarized in Table 9-35.

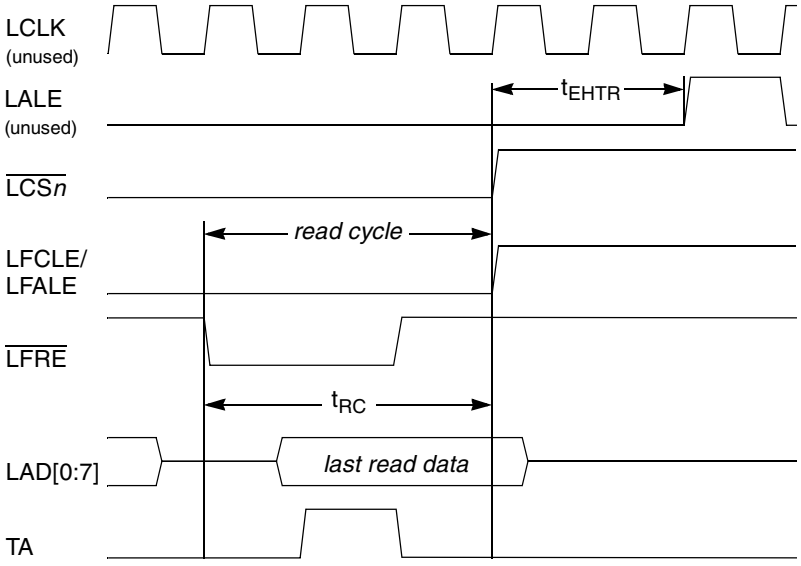
Table 9-35. FCM Read Data Timing Parameters

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.

9.4.3.3.5 FCM Extended Read Hold Timing

Allowance for slow output driver turn-off when reading NAND Flash EEPROMs is made via setting of OR*n*[EHTR] and OR*n*[TRLX]. The extended read data hold time, shown at t_{EHTR} in Figure 9-44 and Figure 9-56, is a delay inserted by FCM between the last data read and another eLBC memory access (requiring LALE assertion). $\overline{\text{LCS}n}$ is negated during t_{EHTR} to allow external devices and bus transceivers time to disable their drivers.



Notes: t_{RC} = Read data cycle time.
t_{EHTR} = Extended read data hold time.

Figure 9-56. FCM Read Data Timing with Extended Hold Time (for TRLX = 0, EHTR = 1, RST = 1, SCY = 1)

9.4.3.4 FCM 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, 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.

9.4.3.4.1 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. 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 9-36 describes the initial values of the boot bank in the memory controller.

Table 9-36. Boot Bank Field Values after Reset for FCM as Boot Controller

Register	Field	Setting
BR0	BA PS DECC WP MSEL ATOM V	0000_0000_0000_0000_0 From cfg_rom_loc From cfg_elbc_ecc 0 001 00 0
OR0	AM BCTLD PGS CSCT CST CHT RST SCY TRLX EHTR	0000_0000_0000_0000_0 0 From cfg_rom_loc 1 1 1 1 From por_cfg_scy[1:3] 1 1

9.4.3.4.2 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 HRESET. 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 9-57.

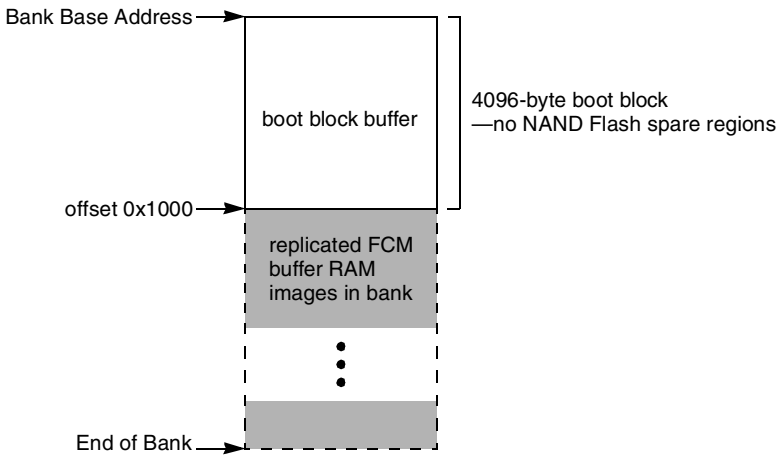


Figure 9-57. FCM Buffer RAM Memory Map During Boot Loading

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 or 16-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 (8-bit port size) or byte offset 11 (16-bit port size).
   - For large-page devices, BI is a single byte read from spare region byte offset 0 (8-bit port size), or two bytes read from spare region byte offsets 0 and 1 (16-bit port size).

   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. The FCM optionally performs ECC checking at boot time depending on the configuration selected during reset. 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 *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.

9.4.4 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 9-58 shows the basic operation of each UPM.

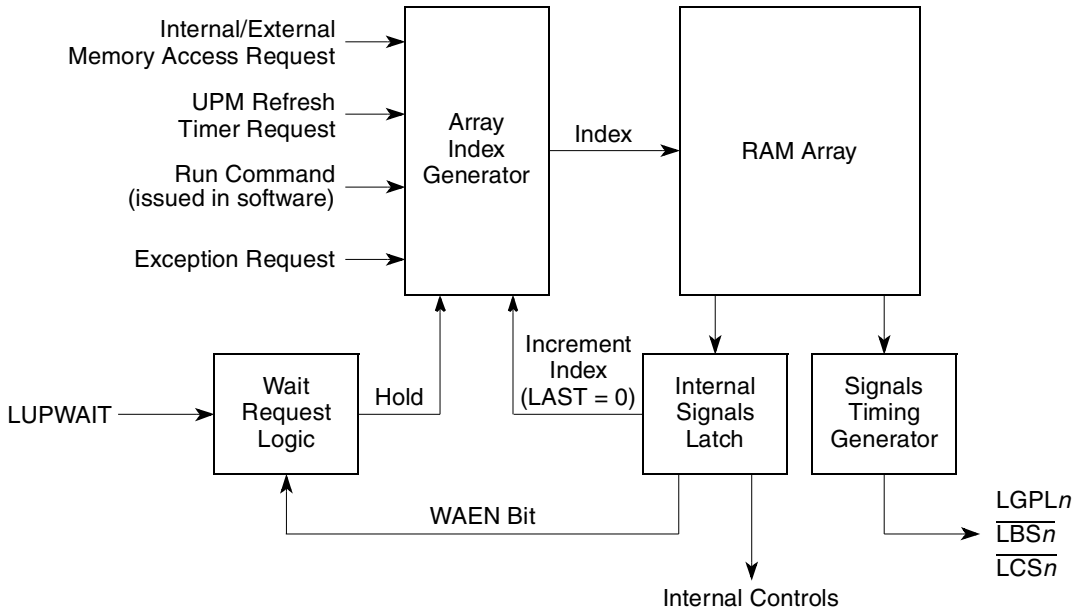


Figure 9-58. User-Programmable Machine Functional Block Diagram

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.

9.4.4.1 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 9-59 and Table 9-37 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.

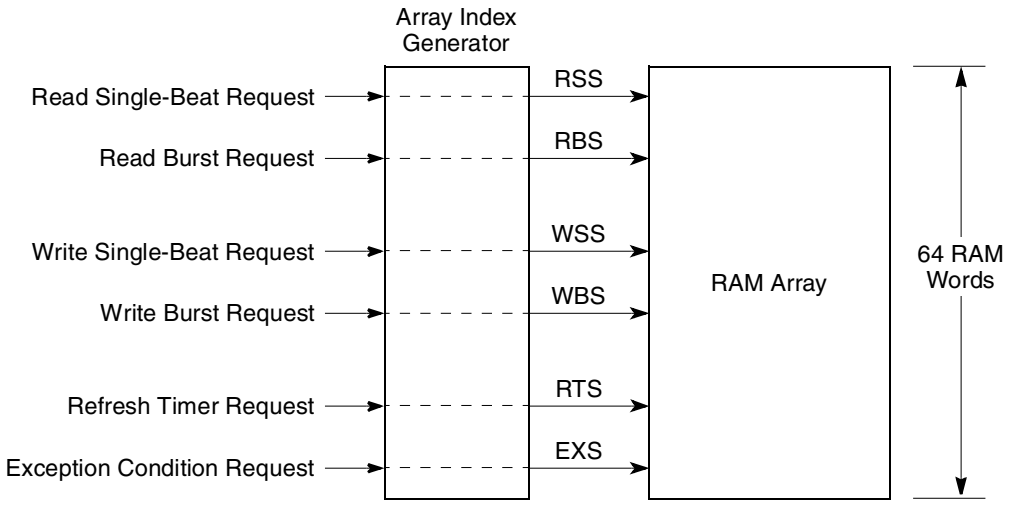


Figure 9-59. RAM Array Indexing

Table 9-37. 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

9.4.4.1.1 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.

9.4.4.1.2 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 9-60 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.

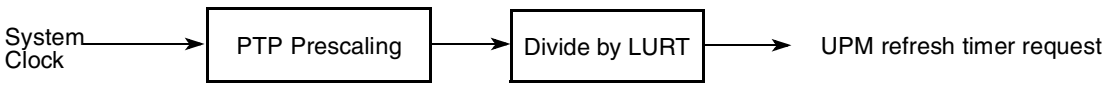


Figure 9-60. Memory Refresh Timer Request Block Diagram

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.

9.4.4.1.3 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.

9.4.4.1.4 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.

9.4.4.2 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

MxMR/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 MxMR/MDR register and the dummy accesses to the UPM memory region, two rules must be followed:

- Since the result of any update to the MxMR/MDR register must be in effect before the dummy read or write to the UPM region, a write to MxMR/MDR should be followed immediately by a read of MxMR/MDR.
- The UPM memory region should have the same MMU settings as the memory region containing the MxMR 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 MxMR. 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.

9.4.4.2.1 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 MxMR 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 MxMR[MAD]. If incremented, the previous dummy write transaction is completed; proceed to step 6. Repeat step 5 until incremented.
6. Program MxMR for the second write with the desired RAM array address.
7. Write pattern/data to MDR to ensure that the MxMR 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 steps 1 and 2 or steps 6 and 7 are reversed, step 3 or 8 (as appropriate) is replaced by the following:

- Read M*x*MR to ensure that the M*x*MR has already been updated with the desired configuration.

9.4.4.2.2 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.

9.4.4.3 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. 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.

The division of UPM bus cycles into phases is shown in Figure 9-61.

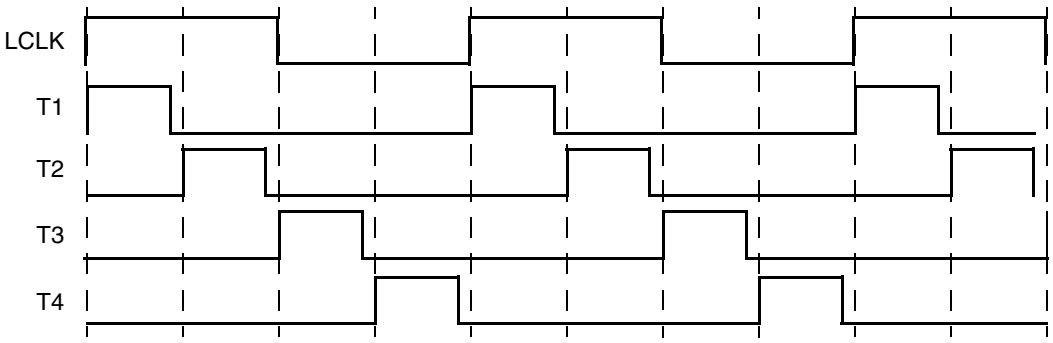


Figure 9-61. UPM Clock Scheme

9.4.4.4 RAM Array

The RAM array for each UPM is 64 locations deep and 32 bits wide, as shown in Figure 9-62. 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.

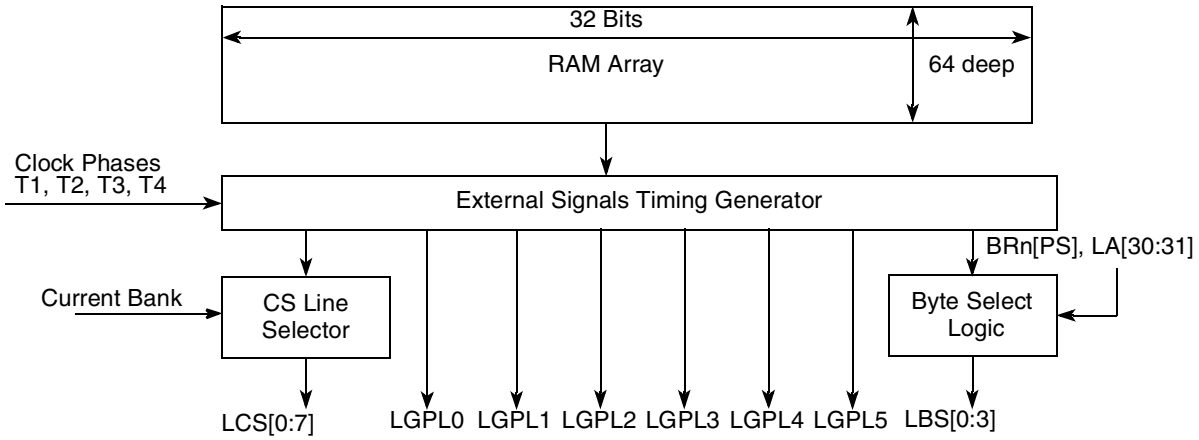


Figure 9-62. RAM Array and Signal Generation

9.4.4.4.1 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 9-35 shows the RAM word fields. he 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.

	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 9-63. RAM Word Fields

Table 9-38 contains descriptions of the RAM word fields.

Table 9-38. 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.
1	CST2	Chip select timing 2. Defines the state (0 or 1) of $\overline{\text{LCS}}n$ during bus clock quarter phase 2.
2	CST3	Chip select timing 3. Defines the state (0 or 1) of $\overline{\text{LCS}}n$ during bus clock quarter phase 3.
3	CST4	Chip select timing 4. Defines the state (0 or 1) of $\overline{\text{LCS}}n$ during bus clock quarter phase 4.
4	BST1	Byte select timing 1. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 1.
5	BST2	Byte select timing 2. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 2.
6	BST3	Byte select timing 3. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 3.
7	BST4	Byte select timing 4. Defines the state (0 or 1) of $\overline{\text{LBS}}$ during bus clock quarter phase 4.
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)

Table 9-38. RAM Word Field Descriptions (continued)

Bits	Name	Description
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).
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

Table 9-38. RAM Word Field Descriptions (continued)

Bits	Name	Description
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.
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 9.4.4.4.7, “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:** Source ID debug mode is only supported for the AMX = 00 setting. **Note:** AMX must not change values in any RAM word which begins a loop.
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.

Table 9-38. RAM Word Field Descriptions (continued)

Bits	Name	Description
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.

9.4.4.4.2 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 9-64 shows how UPMs control $\overline{\text{LCS}n}$ signals.

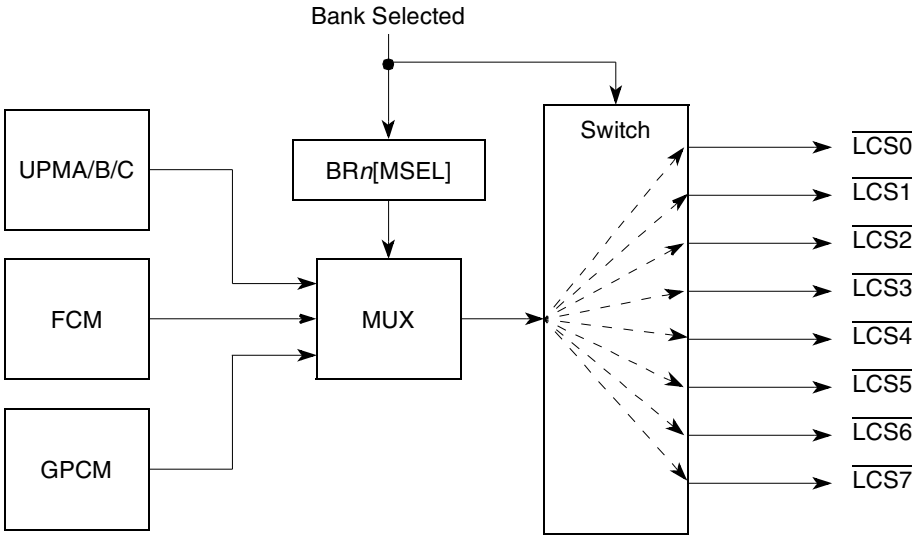


Figure 9-64. $\overline{\text{LCS}n}$ Signal Selection

9.4.4.4.3 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 $\overline{\text{LBS}}$[0:3] signal. The timing of all four byte-select signals is specified in the RAM word. However, $\overline{\text{LBS}}$[0:3] are also controlled by the port size of the accessed bank, the number of bytes to transfer, and the address accessed. Figure 9-65 shows how UPMs control $\overline{\text{LBS}}$[0:3].

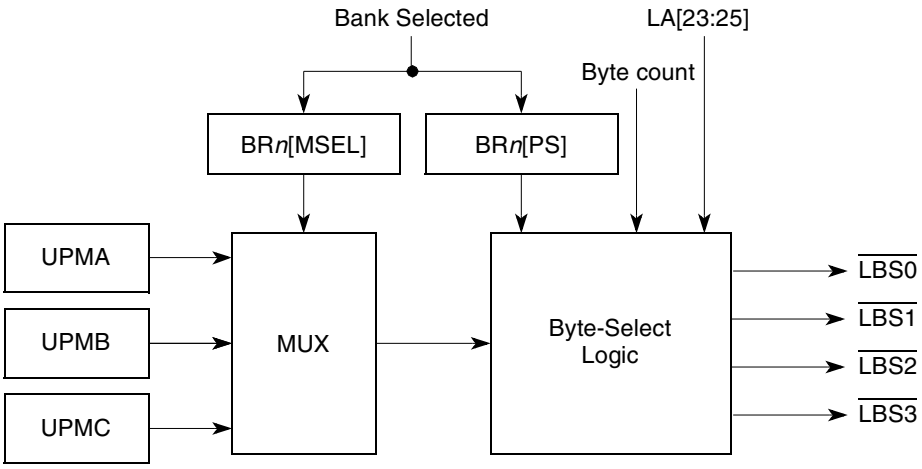


Figure 9-65. $\overline{\text{LBS}}$ Signal Selection

The uppermost byte select ($\overline{\text{LBS0}}$), when asserted, indicates that LAD[0:7] contains valid data during a cycle. Likewise, $\overline{\text{LBS1}}$ indicates that LAD[8:15] contain valid data, $\overline{\text{LBS2}}$ indicates that LAD[16:23] contains valid data, and $\overline{\text{LBS3}}$ indicates that LAD[24:31] contain valid data. For a UPM refresh timer request, all $\overline{\text{LBS}}$[0:3] signals are asserted/negated by the UPM according to the refresh pattern only. Following any internal bus monitor exception, the $\overline{\text{LBS}}$[0:3] signals are negated regardless of the exception handling provided by any UPM exception pattern to prevent spurious writes to external RAM.

9.4.4.4.4 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.

9.4.4.4.5 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 9-39. 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 9-39. 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

9.4.4.4.6 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.

9.4.4.4.7 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 9-40 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/LAD[16:31] signals. When AMX = 10, LAD[0:15] are driven low during an address phase.

NOTE: IPG Defect 1253

Multiple-bank DRAM and SDRAM devices require that the bank address be driven during both RAS and CAS cycles. The UPM does not support a persistent bank address on both RAS and CAS cycles. Therefore, external logic must be used to supply a bank address to these devices.

Table 9-40. 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												
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.

9.4.4.4.8 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 9-66 shows how data sampling is controlled by the UPM.

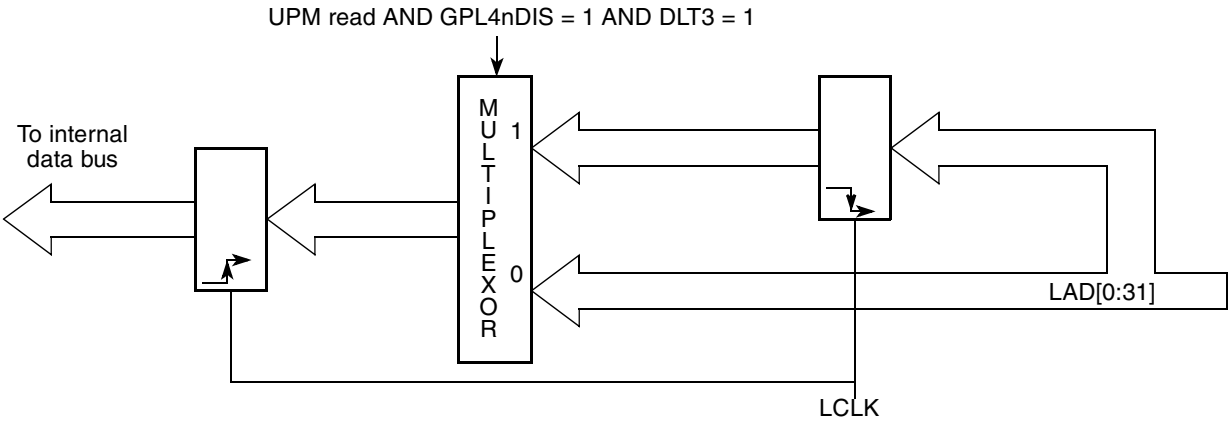


Figure 9-66. UPM Read Access Data Sampling

9.4.4.4.9 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.

9.4.4.4.10 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 9-67 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.

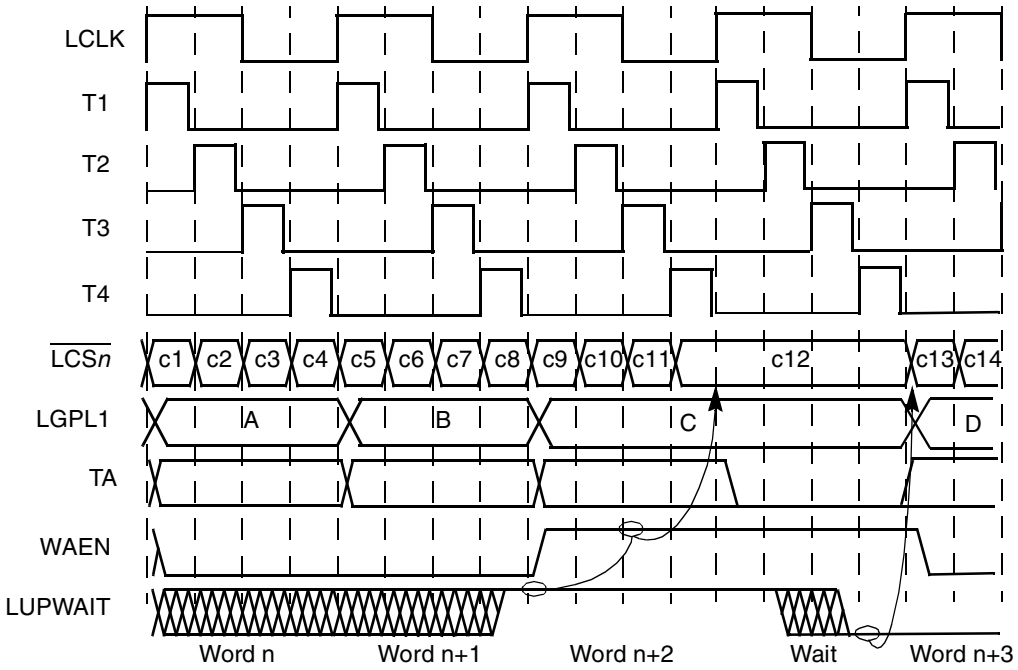


Figure 9-67. Effect of LUPWAIT Signal

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.

9.4.4.5 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.

9.5 Initialization/Application Information

This section provides information about the following:

9.5.1 Interfacing to Peripherals in Different Address Modes

This section provides guidelines for interfacing to peripherals.

and 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]

9.5.1.1 Multiplexed Address/Data Bus for 26-Bit Addressing

In this mode, the eLBC is used with port sizes of 8 and 16 bits; address bits A[6:21] will appear on LAD[0:15] (with zero bits on LAD[16:31]) during address phases, while the lower 10 bits of the address, A[22:31], are driven permanently on LA[22:31]. The connection is illustrated in Figure 9-68.

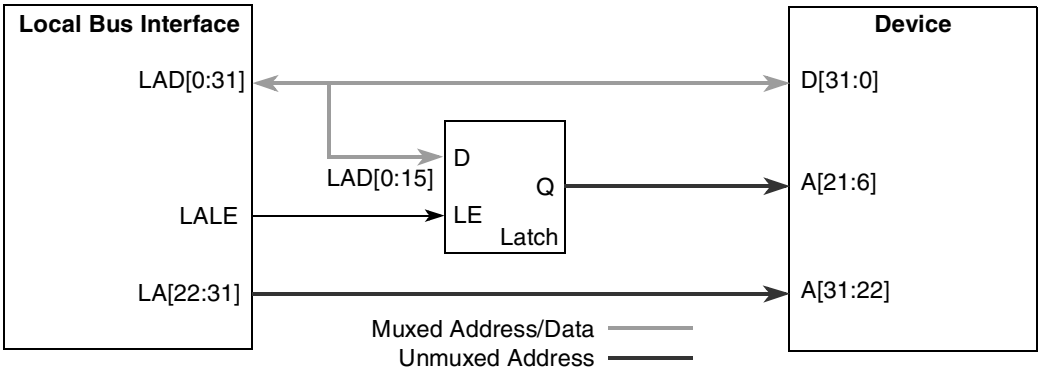


Figure 9-68. Multiplexed Address/Data Bus for 26-Bit Addressing (LBCR[AS16] = 1)

9.5.1.2 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[6:31], in which case addresses driven onto LAD during address phases are simply ignored. The connection is illustrated in Figure 9-69. In non-multiplexed mode, the waveforms

remain the same except for a few things, such as ASHIFT parameter, LAD bus turnaround time, LALE timings, that can be ignored.

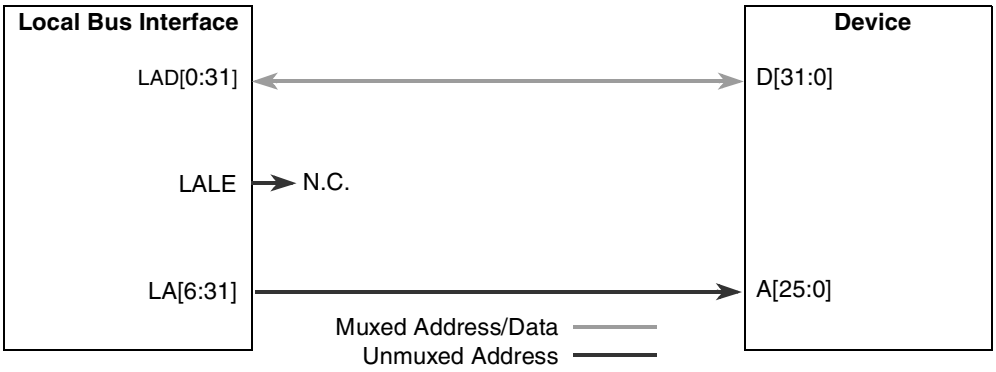


Figure 9-69. Non-Multiplexed Address and Data Buses

9.5.1.3 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 9-70 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.

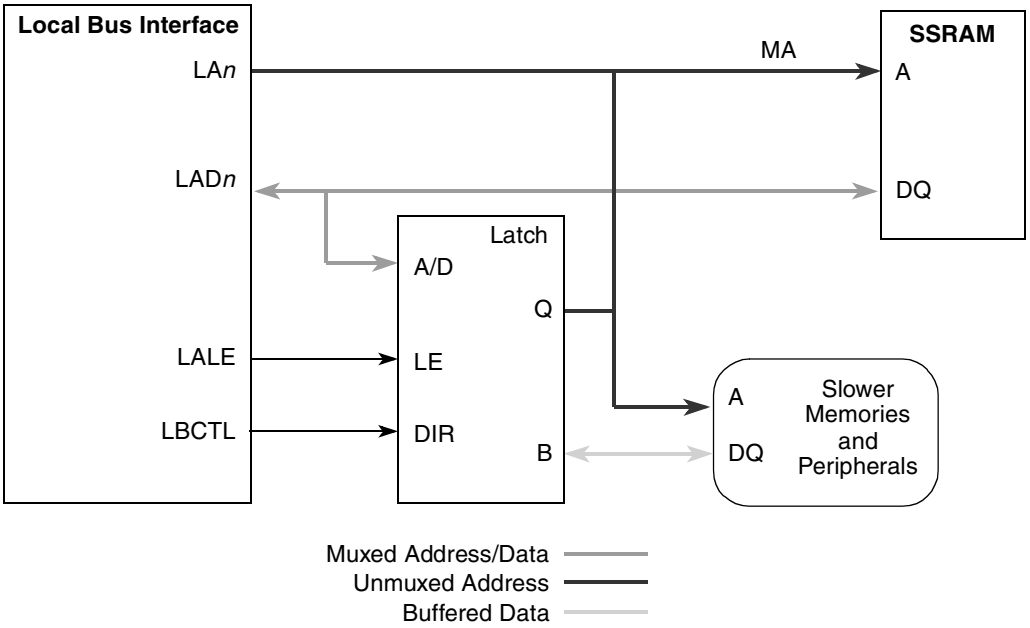


Figure 9-70. Local Bus Peripheral Hierarchy for High Bus Speeds

9.5.1.4 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 9-71 illustrates GPCM address timings.

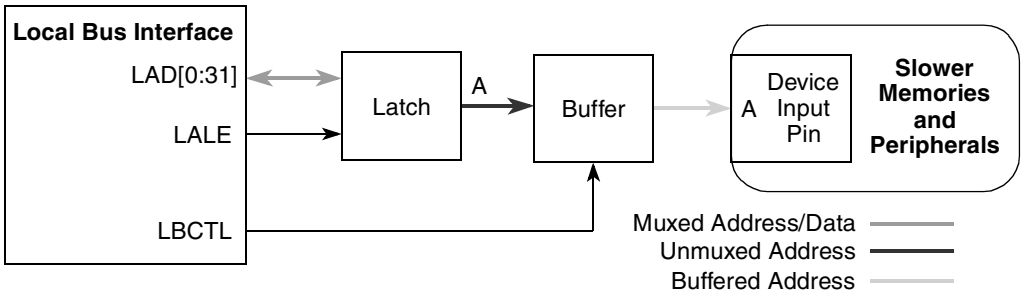


Figure 9-71. GPCM Address Timings

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 to 6 ns, so for a 100-MHz bus frequency, $\overline{\text{LCS}}$ should arrive on the order of 2 to 3 bus clocks later.

For data timings, only the propagation delay of one buffer plus the actual data setup time has to be considered. See Figure 9-72 for an illustration of GPCM data timings.

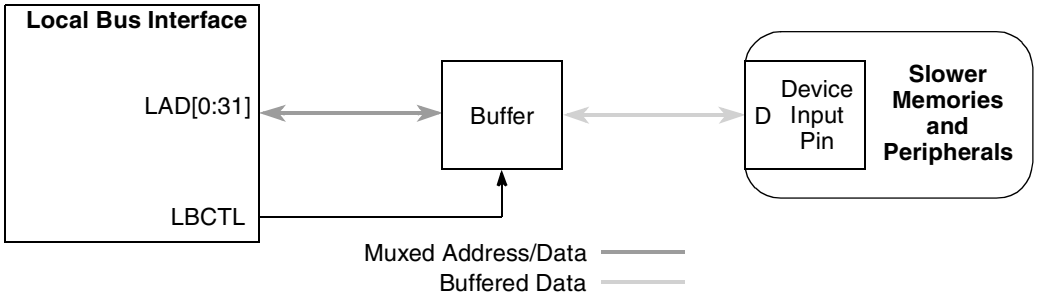


Figure 9-72. GPCM Data Timings

9.5.2 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

9.5.2.1 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.

9.5.2.2 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.

9.5.2.3 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.

9.5.2.4 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.

9.5.3 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 9-73 shows the device connections on the data bus.

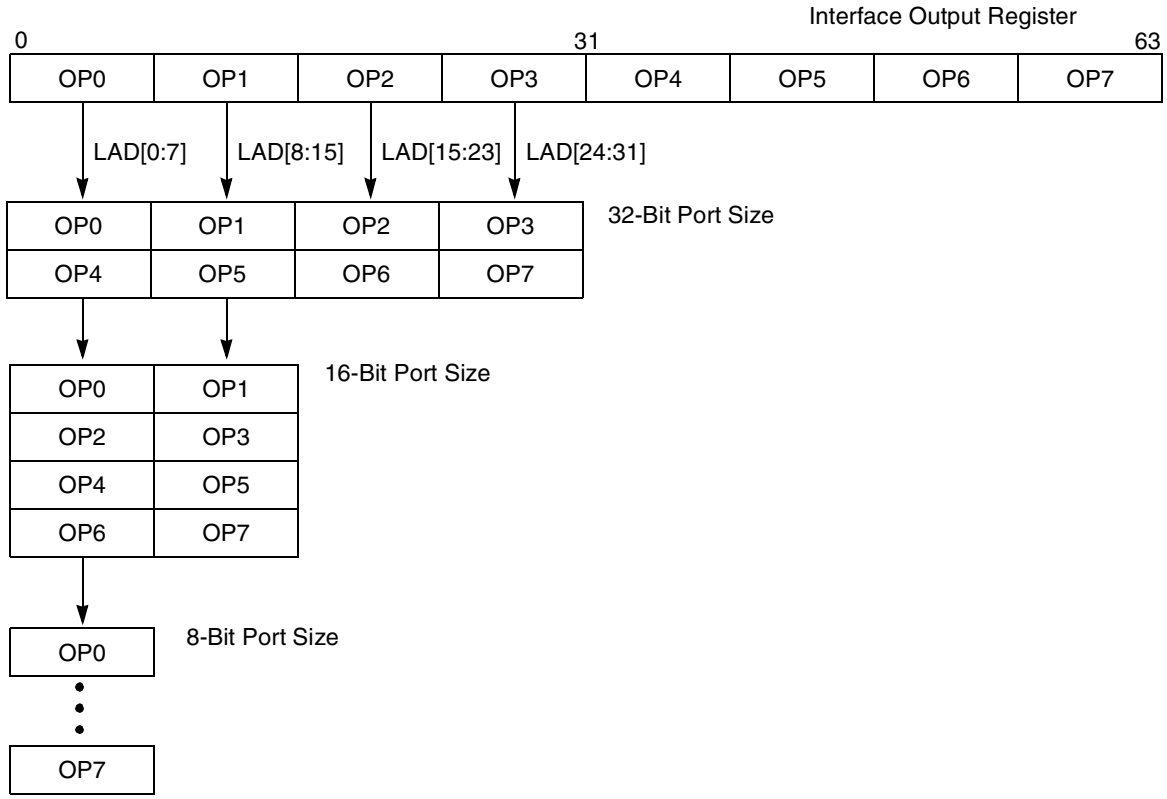


Figure 9-73. Interface to Different Port-Size Devices

Table 9-41 lists the bytes required on the data bus for read cycles.

Table 9-41. 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 9-41. 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.

9.5.4 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.

9.5.4.1 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 9-42. 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 9-42. FCM Register Settings for Soft Reset (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0xFF00_0000	CMD0 = 0xFF = reset command; other commands unused
FBAR	—	Unused

Table 9-42. FCM Register Settings for Soft Reset (OR*n*[PGS] = 1) (continued)

Register	Initial Contents	Description
FPAR	—	Unused
FBCR	—	Unused
MDR	—	Unused
FIR	0x4000_0000	OP0 = CM0 = command 0; OP1–OP7 = NOP

9.5.4.2 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 9-43. This sequence does not require use of the shared FCM buffer RAM, but reads the NAND Flash status into register MDR[AS0] for 8-bit devices. 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 9-43. FCM Register Settings for Status Read (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x7000_0000	CMD0 = 0x70 = read status command; other commands unused
FBAR	—	Unused
FPAR	—	Unused
FBCR	—	Unused
MDR	—	Status returned in AS0
FIR	0x4B00_0000	OP0 = CM0 = command 0; OP1 = RS = read status to MDR; OP2–OP7 = NOP

9.5.4.3 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 9-44. 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 9-44. FCM Register Settings for ID Read (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x9000_0000	CMD0 = 0x90 = read ID command; other commands unused
FBAR	—	Unused
FPAR	—	Unused

Table 9-44. FCM Register Settings for ID Read (OR*n*[PGS] = 1) (continued)

Register	Initial Contents	Description
FBCR	—	Unused
MDR	All zeros	AS0 = 0x00 = dummy address for read ID command; AS0–AS3 return with first 4 bytes of ID code
FIR	0x43BB_BBB0	OP0 = CM0 = command 0; OP1 = UA = user address from MDR; OP2–OP6 = RS = read 4 bytes ID into MDR[AS3–AS0]; OP7 = NOP

9.5.4.4 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 9-45. 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 9-45. FCM Register Settings for Page Read (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x0030_0000	CMD0 = 0x00 = random read address entry; CMD1 = 0x30 = read page
FBAR	block index (for example block 0x0001_0AB4)	BLK locates index of 128-Kbyte block
FPAR	page offset (for example 0x0000_5000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; MS = 0 and CI = 0
FBCR	All zeros	BC = 0 to read entire 2112-byte page with ECC check
MDR	—	Unused
FIR	0x4125_E000	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

9.5.4.5 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 9-46. 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 9-46. FCM Register Settings for Block Erase (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x6070_D000	CMD0 = 0x60 = block address entry; CMD1 = 0x70 = read status CMD2 = 0xD0 = erase block;
FBAR	Block Index (for example block 0x0001_0AB4)	BLK locates index of 128-Kbyte block
FPAR	All zeros	PI = 0 to locate block boundary
FBCR	—	Unused
MDR	—	Returns with AS0 holding erase status
FIR	0x426D_B000	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

9.5.4.6 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 9-47. 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

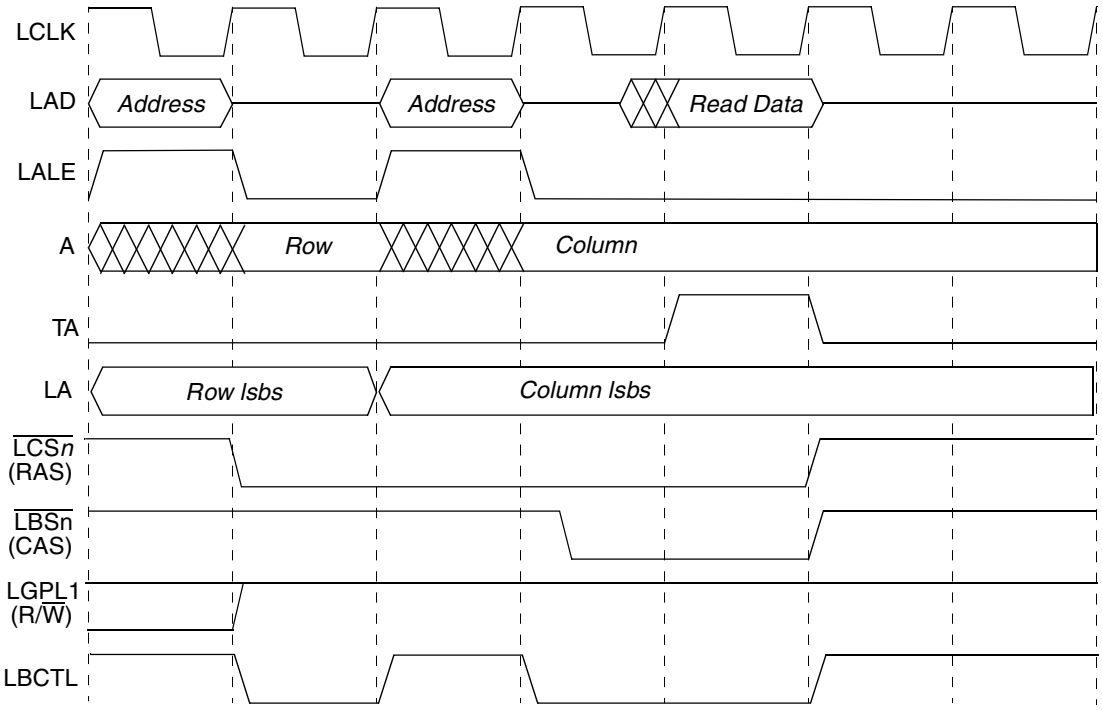
happen that a new command is issued to the NAND Flash device even when the device has not yet finished processing the previous request. This may also result in unpredictable behavior.

Table 9-47. FCM Register Settings for Page Program (OR*n*[PGS] = 1)

Register	Initial Contents	Description
FCR	0x8070_1000	CMD0 = 0x80 = page address and data entry; CMD1 = 0x70 = read status CMD2 = 0x10 = program page;
FBAR	block index (for example block 0x0001_0AB4)	BLK locates index of 128-Kbyte block
FPAR	page offset (for example 0x0000_5000 locates page 5, buffer 1)	PI locates page index in BLK; PI mod 2 indexes FCM buffer RAM; MS = 0 and CI = 0
FBCR	All zeros	BC = 0 to write entire 2112-Byte page with ECC generation
MDR	—	Returns with AS0 holding program status
FIR	0x4128_6DB0	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

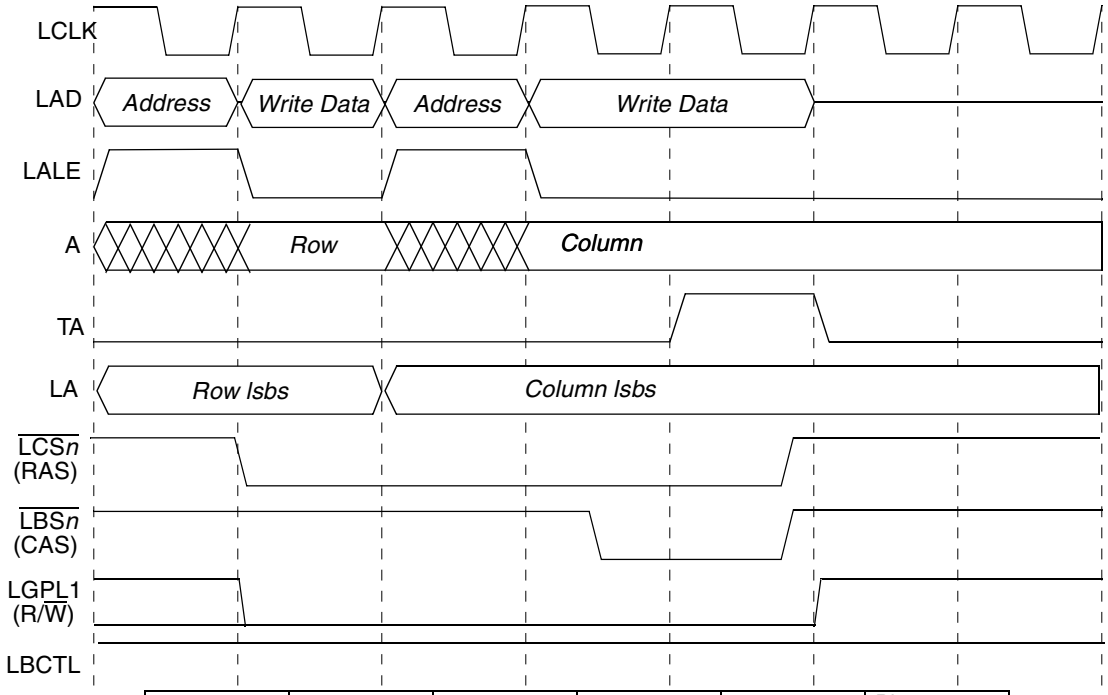
9.5.5 Interfacing to Fast-Page Mode DRAM Using UPM

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 describes timing diagrams for various UPM configurations for fast-page mode DRAM, with LCRR[CLKDIV] = 4 (clock ratio of 8) or 8 (clock ratio of 16). The illustrative examples shown in Figure 9-74–Figure 9-79 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.



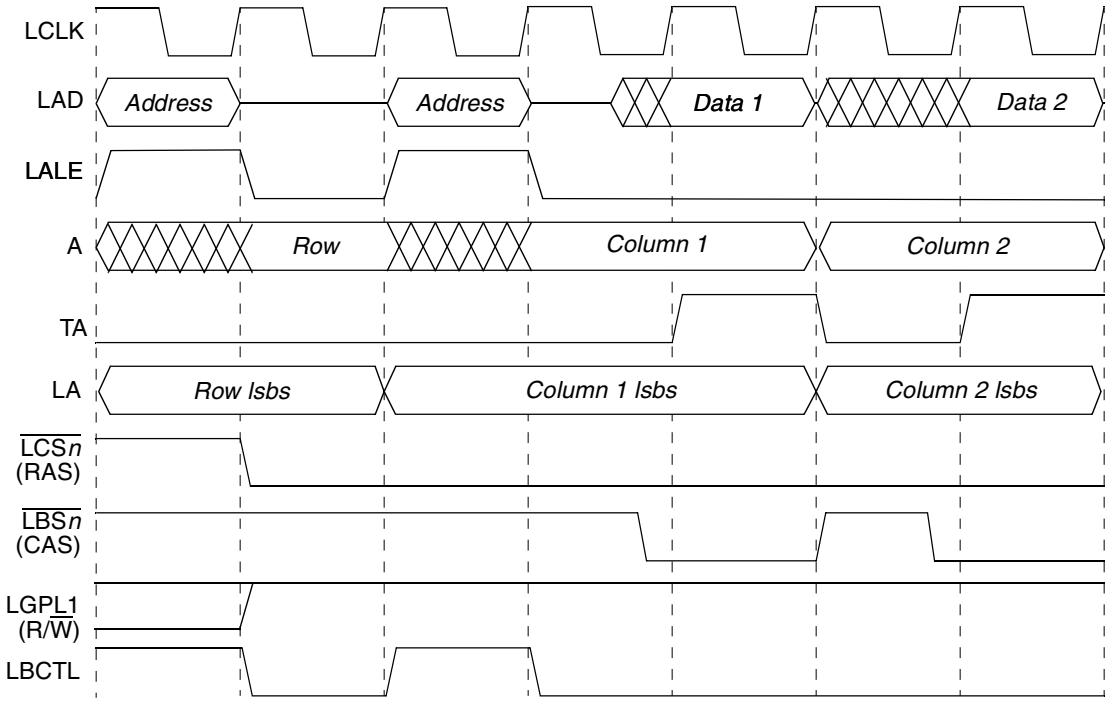
cst1 cst2 cst3 cst4	0 0 0 0	LALE pause (due to change in AMX)	0 0 0 0	0 0 0 0	Bit 0 Bit 1 Bit 2 Bit 3
bst1 bst2 bst3 bst4	1 1 1 1		1 0 0 0	0 0 0 0	Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1					Bit 8 Bit 9 Bit 10 Bit 11
g1t1 g1t3	1 1		1 1	1 1	Bit 12 Bit 13
g2t1 g2t3					Bit 14 Bit 15
g3t1 g3t3					Bit 16 Bit 17
g4t1 g4t3					Bit 18 Bit 19
g5t1 g5t3					Bit 20 Bit 21
redo[0] redo[1]					Bit 22 Bit 23
loop exen	0 0		0 0	0 0	Bit 24 Bit 25
amx0 amx1 na uta todt last	1 0 0 0 0 0		0 0 0 0 0 0	0 0 0 1 1 1	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	RSS	**RSS + 1**	**RSS + 1**	**RSS + 2**	

Figure 9-74. Single-Beat Read Access to FPM DRAM



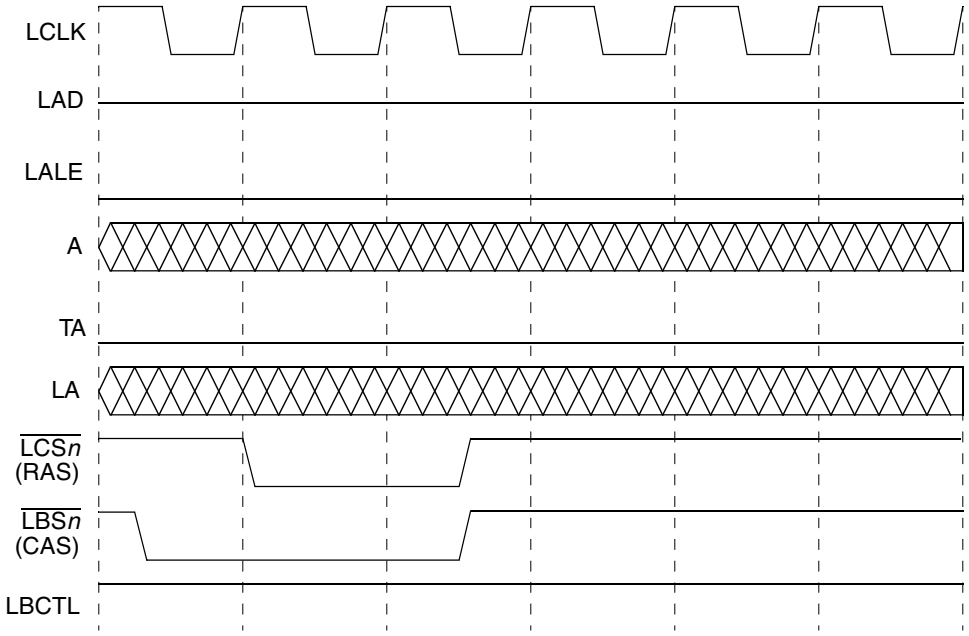
cst1 cst2 cst3 cst4	0 0 0 0	LALE pause (due to change in AMX)	0 0 0 0	0 0 0 1	Bit 0 Bit 1 Bit 2 Bit 3
bst1 bst2 bst3 bst4	1 1 1 1		1 1 0 0	0 0 0 1	Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1					Bit 8 Bit 9 Bit 10 Bit 11
g1t1 g1t3	0 0		0 0	0 0	Bit 12 Bit 13
g2t1 g2t3					Bit 14 Bit 15
g3t1 g3t3					Bit 16 Bit 17
g4t1 g4t3					Bit 18 Bit 19
g5t1 g5t3					Bit 20 Bit 21
redo[0] redo[1]					Bit 22 Bit 23
loop exen	0 0		0 0	0 0	Bit 24 Bit 25
amx0 amx1 na uta todt last	1 0 0 0 0 0		0 0 0 0 0 0	0 0 0 1 1 1	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	WSS	**WSS + 1**	**WSS + 1**	**WSS + 2**	

Figure 9-75. Single-Beat Write Access to FPM DRAM



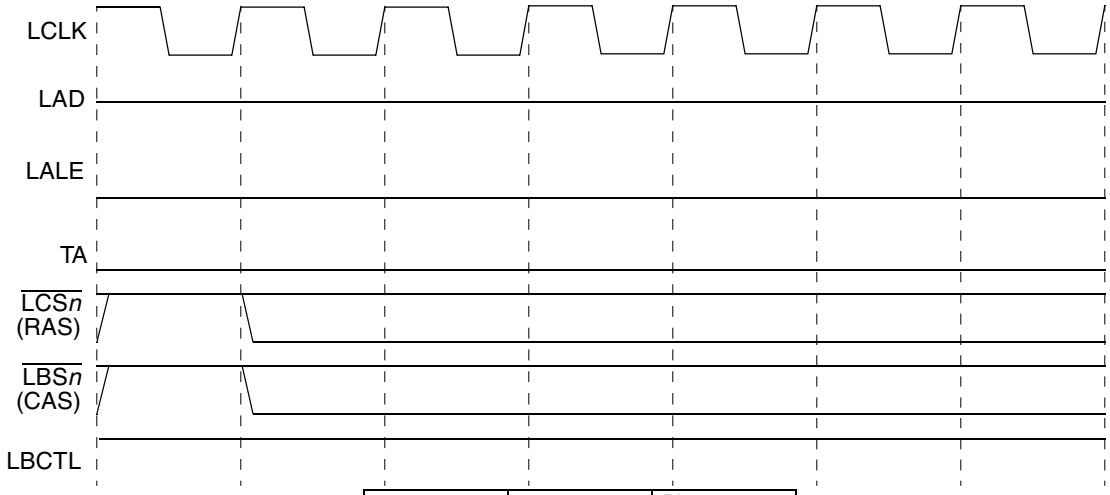
cst1	0	LALE pause (due to change in AMX)	0	0	1	Bit 0
cst2	0		0	0	1	Bit 1
cst3	0		0	0	1	Bit 2
cst4	0		0	0	1	Bit 3
bst1	1		1	0	1	Bit 4
bst2	1		1	0	1	Bit 5
bst3	1		1	0	1	Bit 6
bst4	1		0	0	1	Bit 7
g0l0						Bit 8
g0l1						Bit 9
g0h0						Bit 10
g0h1						Bit 11
g1t1	1		1	1	1	Bit 12
g1t3	1		1	1	1	Bit 13
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 9-76. Burst Read Access to FPM DRAM Using LOOP (Two Beats Shown)



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
	PTS	**PTS + 1**	**PTS + 2**	

Figure 9-77. Refresh Cycle (CBR) to FPM DRAM



cst1 cst2 cst3 cst4	1 1 1 1	Bit 0 Bit 1 Bit 2 Bit 3
bst1 bst2 bst3 bst4	1 1 1 1	Bit 4 Bit 5 Bit 6 Bit 7
g0l0 g0l1 g0h0 g0h1		Bit 8 Bit 9 Bit 10 Bit 11
g1t1 g1t3		Bit 12 Bit 13
g2t1 g2t3		Bit 14 Bit 15
g3t1 g3t3		Bit 16 Bit 17
g4t1 g4t3		Bit 18 Bit 19
g5t1 g5t3		Bit 20 Bit 21
redo[0] redo[1]		Bit 22 Bit 23
loop exen	0 0	Bit 24 Bit 25
amx0 amx1 na uta todt last	0 0 0 0 1 1	Bit 26 Bit 27 Bit 28 Bit 29 Bit 30 Bit 31
	EXS	

Figure 9-78. Exception Cycle

9.5.6 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 9-79 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. As ZBT SRAMs are

mostly used by performance-critical applications, it is assumed here that, typically, the maximum width of the local bus of 32 bits will be used.

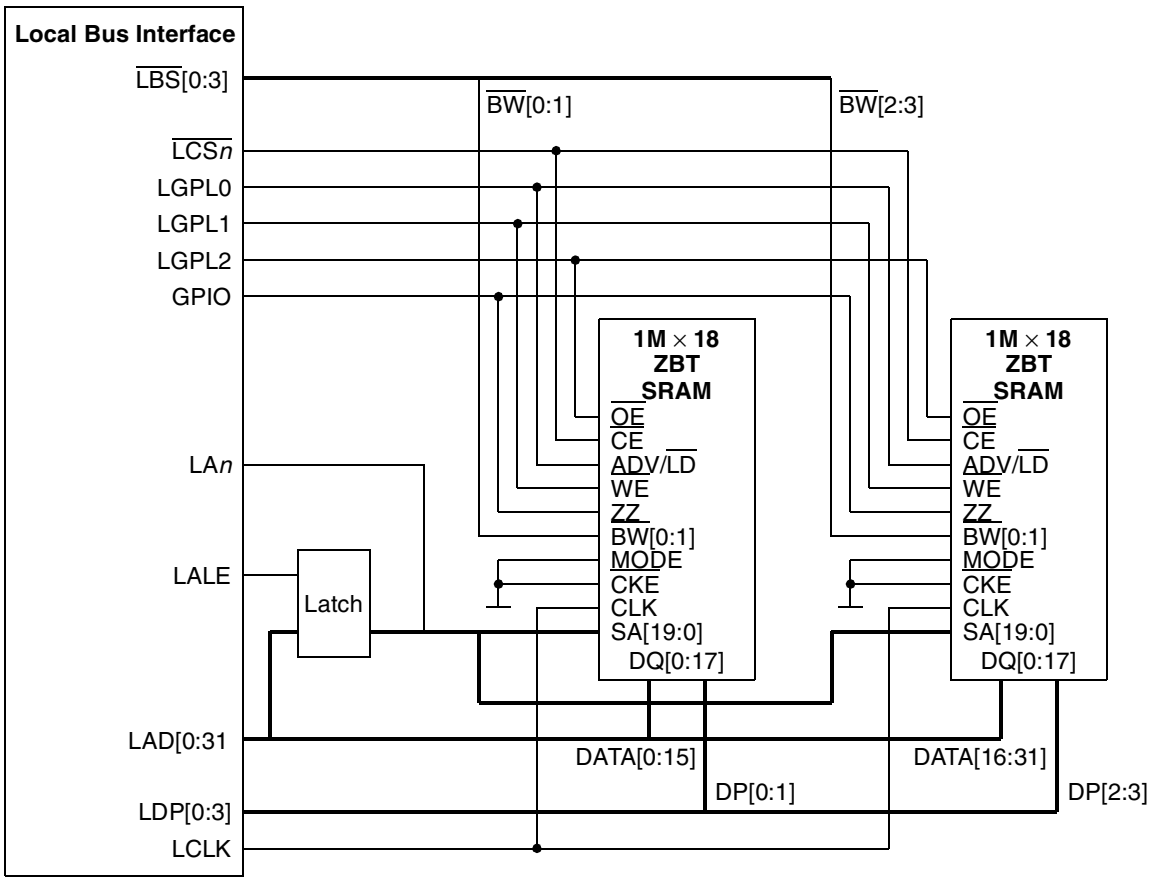


Figure 9-79. Interface to ZBT SRAM

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. $\overline{\text{CKE}}$ 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 {A21, A22} = {0,0} for the first burst, and {A21, A22} = {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 $\overline{\text{WE}}$). The UPM controller basically has to wait for the end of the SRAM burst to avoid bus contention with further bus activities.

Chapter 10
Display Interface Unit (DIU)

This chapter describes the display interface unit (DIU).

10.1 Introduction

The DIU is a display controller designed to handle TFT LCD display. Besides generating all the signals required to drive the display, the DIU handles real time blending of up to three planes onto the display.

10.1.1 Features

- Supports any resolution up to 1280×1024 (single-plane), or any resolution up to 1024×768 (3 multiple planes)
- Display refresh rate: 60 Hz (up to 1280×1024) or 72 Hz (up to 1024×768)
- Display color depth: up to 24 bpp
- Display interfaces: parallel TTL
- Maximum number of input planes: 3 (for display resolutions ≤ 1024 × 768) Input pixel format: RGB and 256 level gray scale.
  Programmable bit order definition up to 8 bits per component.
- Hardware cursor: 32 × 32 pixels, 16 bpp
- α-blending range: up to 256 levels
- Chroma keying: Selectable by range
- Independent programmable gamma adjustments for each color component
- Memory write back mode to store intermediate results, virtually extending the number of graphics planes
- Internal direct memory access (DMA) module: with five channels, to transfer data from or back to memory

10.1.2 Modes of Operation

The DIU has five modes of operation:

Mode 0: DIU OFF. In this mode, the DIU is disabled.

Mode 1: All three planes output to display. This is the typical operating mode of the DIU.

Mode 2: Plane 1 to display, Planes 2 and 3 written back to memory. This mode is used to display a plane while processing (and writing back to memory) the data for other planes.

Mode 3: All three planes written back to memory. This mode is used to process (and write back to memory) the data for the planes without displaying any of them.

Mode 4: Color Bar Generation. This is a debug mode to check the operation of the DIU without the need for setting up the display memory structures in memory.

These modes are set by programming the DIU_MODE register. See Section 10.3.1.8, "DIU_MODE Register," for more information.

10.2 External Signal Description

Table 10-1 describes the DIU input and output signals, the meaning of their different states, and relative timing information for assertion and negation.

NOTE

The DIU_LD[23:0] signals are multiplexed with GPIO1 signals. See Section 3.2.3, "DIU and GPIO1 Signal Multiplexing," for more information. The functionality of these signals is determined by the DIU8 and DIU16 fields in the general-purpose I/O control register (GPIOCR) in the global utilities block. Note that the GPIOCR defaults to GPIO functionality on the DIU signal pins; the GPIOCR must be initialized to the desired DIU signaling for proper operation.

Table 10-1. Display Interface—Detailed Signal Descriptions

Signal	I/O	Description	
pix_clk_out	O	Pixel clock. This signals is used to drive the display panel.	
DIU_VSYNC	O	Vertical synchronizing signal. This signal indicates the beginning of a new frame. Note this signal may alternately be programmed to output a composite sync (CSYNC) signal by programing SYN_POL[BP_VS]. See Section 10.3.1.16, "SYN_POL Register," for more information. The composite sync signal combines the horizontal and vertical synchronizing signals to form a composite synchronizing signal. It includes both the HSYNC pulse and the VSYNC pulse. The default output is DIU_VSYNC.	
		State Meaning	Asserted at the beginning of a new frame.
		Timing	Asserted with the first cycle of the frame period. The length of the pulse is programmable.
DIU_HSYNC	O	Horizontal synchronizing signal. This signal indicates the beginning of a new line. Note this signal may alternately be programmed to output a composite sync (CSYNC) signal by programing SYN_POL[BP_HS]. See Section 10.3.1.16, "SYN_POL Register," for more information. The composite sync signal combines the horizontal and vertical synchronizing signals to form a composite synchronizing signal. It includes both the HSYNC pulse and the VSYNC pulse. The default output is DIU_HSYNC.	
		State Meaning	Asserted at the beginning of a new line.
		Timing	Asserted with the first cycle of a new line. The length of the pulse is program able.
DIU_DE	O	Data enable. This signal qualifies the data on the data output signals (DIU_LD)	
		State Meaning	Deasserted: diu_ld data is not valid Asserted: diu_ld data is valid.

Table 10-1. Display Interface—Detailed Signal Descriptions (continued)

Signal	I/O	Description
DIU_LD[23:0]	O	Data output signals. • DIU_LD[23:16] = Red[7:0]. DIU_LD[23] is the most significant bit, and DIU_LD[16] is the least significant bit of the Red component. • DIU_LD[15:8] = Green[7:0]. DIU_LD[15] is the most significant bit, and DIU_LD[8] is the least significant bit of the Green component. • DIU_LD[7:0] = Blue[7:0]. DIU_LD[7] is the most significant bit, and DIU_LD[0] is the least significant bit of the Blue component.

10.3 Memory Map and Register Definition

Table 10-2 shows the register memory map for the DIU memory controller.

Table 10-2. DIU Memory Map

Offset (Hex)	Register	Access	Reset	Section/Page
Display Interface Unit Registers—Block Base Address 0x2_C000				
0x000	DESC_1 — Pointer to the Area Descriptor of Plane1	R/W	All zeros	10.3.1.1/10-4
0x004	DESC_2 — Pointer to the Area Descriptor of Plane2	R/W	All zeros	10.3.1.2/10-5
0x008	DESC_3 — Pointer to the Area Descriptor of Plane3	R/W	All zeros	10.3.1.3/10-5
0x00c	GAMMA — Pointer to Gamma Table	R/W	All zeros	10.3.1.4/10-6
0x010	PALETTE — Pointer to Palette	R/W	All zeros	10.3.1.5/10-6
0x014	CURSOR — Pointer to Cursor Bitmap	R/W	All zeros	10.3.1.6/10-7
0x018	CURS_POS — Position of the cursor in the display	R/W	All zeros	10.3.1.7/10-7
0x01c	DIU_MODE — DIU Mode of Operation	R/W	All zeros	10.3.1.8/10-8
0x020	BGND — Background Color	R/W	All zeros	10.3.1.9/10-8
0x024	BGND_WB — Background Color in write back Mode	R/W	All zeros	10.3.1.10/10-9
0x028	DISP_SIZE — Display Size	R/W	All zeros	10.3.1.11/10-10
0x02c	WB_SIZE — Write back Plane Size	R/W	All zeros	10.3.1.12/10-10
0x030	WB_MEM_ADDR — Address to Store the write back Plane	R/W	All zeros	10.3.1.13/10-11
0x034	HSYN_PARA — Horizontal synchronization pulse parameters	R/W	All zeros	10.3.1.14/10-11
0x038	VSYN_PARA — Vertical synchronization pulse parameters	R/W	All zeros	10.3.1.15/10-12
0x03c	SYN_POL — Synchronization Signals Polarity	R/W	All zeros	10.3.1.16/10-13
0x040	THRESHOLDS — The Thresholds	R/W	0x0000_7800	10.3.1.17/10-13
0x044	INT_STATUS — Interrupt Status Register	R	All zeros	10.3.1.18/10-14
0x048	INT_MASK — Interrupt Mask Register	R/W	All zeros	10.3.1.19/10-15

Table 10-2. DIU Memory Map (continued)

Offset (Hex)	Register	Access	Reset	Section/Page
0x04c	COLORBAR1 — Color #1 in the Color Bar, Black	R/W	0xFF00_0000	10.3.1.20/10-15
0x050	COLORBAR2 — Color #2 in the Color Bar, Blue	R/W	0xFF00_00FF	10.3.1.20/10-15
0x054	COLORBAR3 — Color #3 in the Color Bar, Cyan	R/W	0xFF00_7F7F	10.3.1.20/10-15
0x058	COLORBAR4 — Color #4 in the Color Bar, Green	R/W	0xFF00_FF00	10.3.1.20/10-15
0x05c	COLORBAR5 — Color #5 in the Color Bar, Yellow	R/W	0xFF7F_7F00	10.3.1.20/10-15
0x060	COLORBAR6 — Color #6 in the Color Bar, Red	R/W	0xFFFF_0000	10.3.1.20/10-15
0x064	COLORBAR7 — Color #7in the Color Bar, Purple	R/W	0xFF7F_007F	10.3.1.20/10-15
0x068	COLORBAR8 — Color #8 in the Color Bar, White	R/W	0xFFFF_FFFF	10.3.1.20/10-15
0x06c	FILLING — Input, output buffer filling status, for debug purpose	R	All zeros	10.3.1.21/10-17
0x070	PLUT — Priority Look Up Table	R/W	All zeros	10.3.1.22/10-18

10.3.1 Register Descriptions

The following sections describe the DIU registers in detail.

10.3.1.1 Plane 1 Area Descriptor Pointer Register (DESC_1)

Figure 10-1 shows the DESC_1 register.

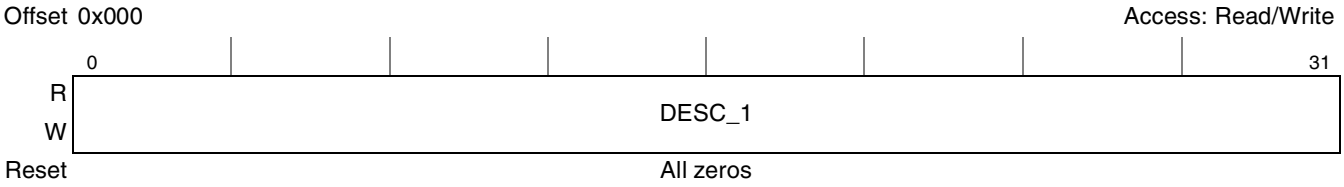


Figure 10-1. Plane 1 Area Descriptor Pointer Register (DESC_1)

Table 10-3 describes the DESC_1 fields.

Table 10-3. DESC_1 Field Descriptions

Bits	Name	Description
0–31	DESC_1	DESC_1 register is the Plane 1 area descriptor pointer. It sets the base address of the first Plane 1 AD (Area Descriptor).This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). DESC_1 = all zeros means no AD available for this plane.

10.3.1.2 Plane 2 Area Descriptor Pointer Register (DESC_2)

Figure 10-2 shows the DESC_2 register.

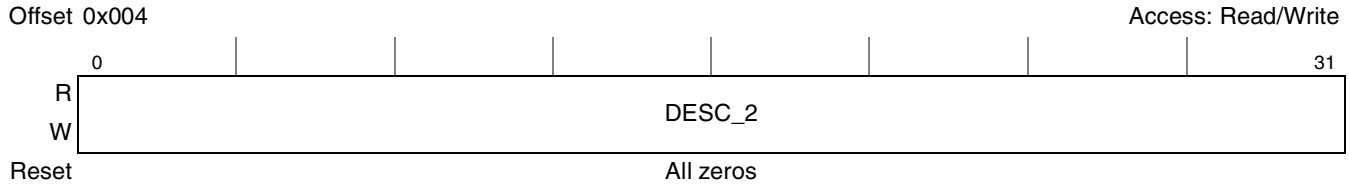


Figure 10-2. Plane 2 Area Descriptor Pointer Register (DESC_2)

Table 10-4 describes the DESC_2 fields.

Table 10-4. DESC_2 Field Descriptions

Bits	Name	Description
0–31	DESC_2	DESC_2 register is the Plane 2 area descriptor pointer. It sets the base address of the first Plane 2 AD.This address must be 64-bit boundary aligned (set the lowest 3 bits to 0).DESC_2 = all zeros means no AD available for this plane.

10.3.1.3 Plane 3 Area Descriptor Pointer Register (DESC_3)

Figure 10-3 shows the DESC_3 register.

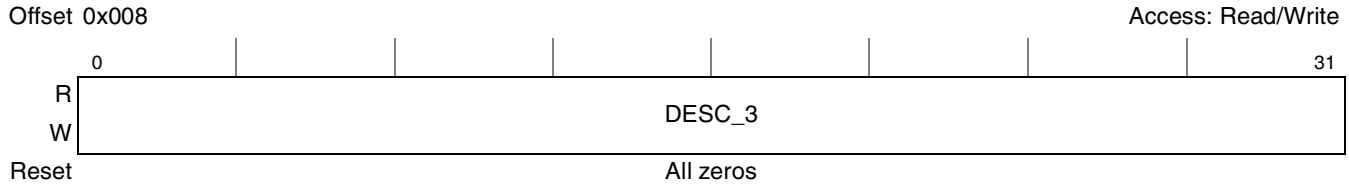


Figure 10-3. Plane 3 Area Descriptor Pointer Register (DESC_3)

Table 10-5 describes the DESC_3 fields.

Table 10-5. DESC_3 Field Descriptions

Bits	Name	Description
0–31	DESC_3	DESC_3 register is the Plane 3 area descriptor pointer. It sets the base address of the first Plane 3 AD. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0). DESC_3 = all zeros means no AD available for this plane.

10.3.1.4 GAMMA Register

Figure 10-4 shows the GAMMA register.

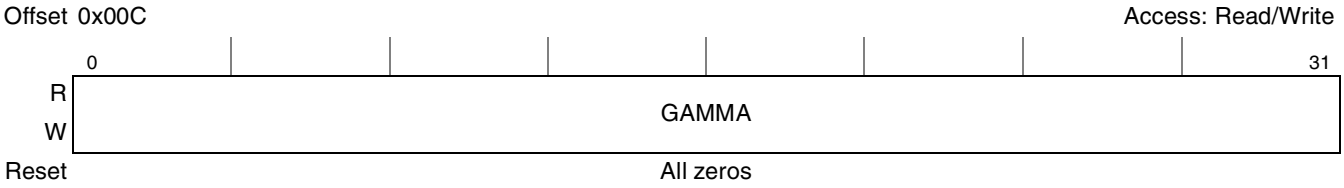


Figure 10-4. GAMMA Register

Table 10-6 describes the GAMMA fields.

Table 10-6. GAMMA Field Descriptions

Bits	Name	Description
0–31	GAMMA	GAMMA register sets the base address to the Gamma table in memory. Writing to this register will cause the DIU to load the new Gamma table from there. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0); 32-byte boundary aligned address is more efficient.Note that when GAMMA=0x0, no Gamma is available.

10.3.1.5 PALETTE Register

Figure 10-5 shows the PALETTE register.

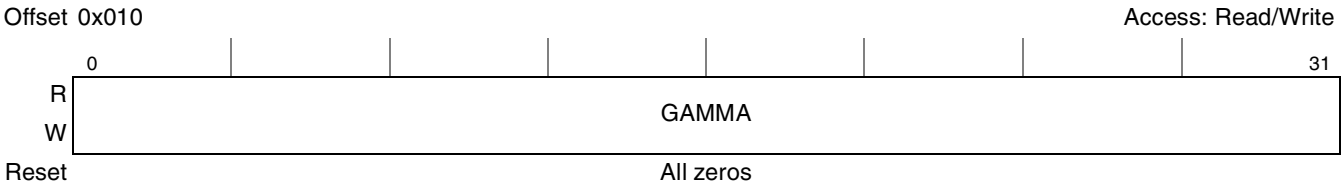


Figure 10-5. PALETTE Register

Table 10-7 describes the PALETTE fields.

Table 10-7. PALETTE Field Descriptions

Bits	Name	Description
0–31	PALETTE	PALETTE register sets the base address to the Palette table in memory. Writing to this register will cause the DIU to load the new Palette table from there. This address must be aligned to an 8-byte boundary (the lowest 3 address bits are 0); however, a 32-byte aligned address is more efficient. Note that palette mode is only supported for display frames; it is not supported for frames used for write-back mode. Also, when PALETTE=0x0, no Palette is available.

10.3.1.6 CURSOR Register

Figure 10-6 shows the CURSOR register.

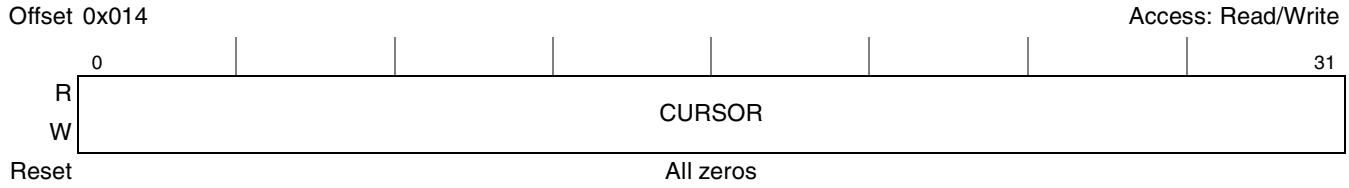


Figure 10-6. CURSOR Register

Table 10-8 describes the CURSOR fields.

Table 10-8. CURSOR Field Descriptions

Bits	Name	Description
0–31	CURSOR	CURSOR register sets the base address to the Cursor bitmap in memory.Writing to this register will cause the DIU to load the new Cursor bitmap from there. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0); 32-byte boundary aligned address is more efficient.

10.3.1.7 CURS_POS Register

The CUR_POS register, shown in Figure 10-7, sets the position of the cursor in the display.

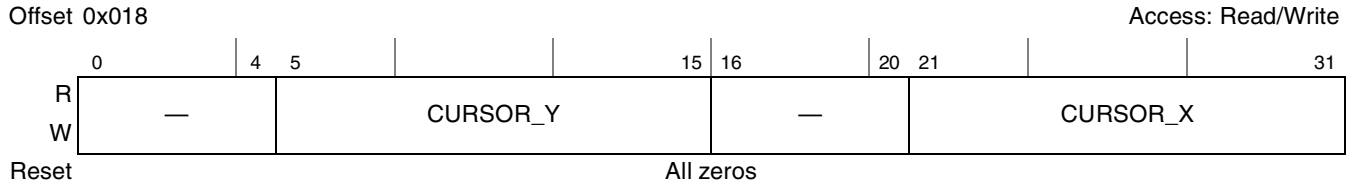


Figure 10-7. CURS_POS Register

Table 10-9 describes the CURS_POS fields.

Table 10-9. CURS_POS Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–15	CURSOR_Y	Vertical position of the cursor (in pixels), top-left corner.
16–20	—	Reserved
21–31	CURSOR_X	Horizontal position of the cursor (in pixels), top-left corner.

NOTE

Programming the CURS_POS register at the middle of a frame changes the cursor position immediately, so one should reprogram it after a VSYNC interrupt is detected.

10.3.1.8 DIU_MODE Register

The DIU_MODE register, shown in Figure 10-8, sets the operation mode of the block.

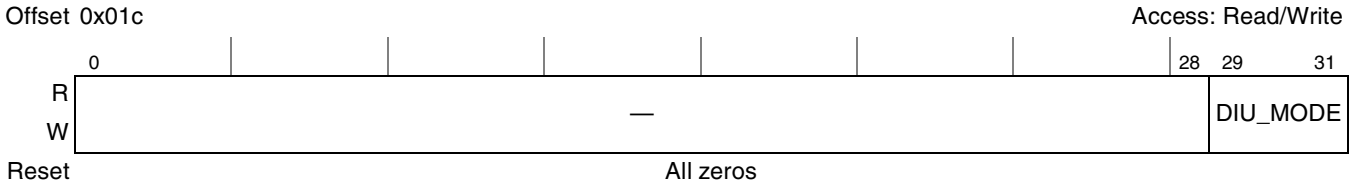


Figure 10-8. DIU_MODE Register

See Table 10-10 for DIU_MODE field descriptions.

Table 10-10. DIU_MODE Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29–31	DIU_MODE	DIU Operation Mode 000 Mode 0: DIU off. 001 Mode 1: All three planes output to display. 010 Mode 2: Plane 1 to display, Planes 2 and 3 written back to memory. 011 Mode 3: All three planes written back to memory. 100 Mode 4: Color bar generation. All other encodings are reserved[1] **Note:** The DIU must not be enabled (DIU_MODE ≠ 000) if the core is in a boot hold off state. Software can determined the boot holdoff state of the core by polling the PORT0_EN bit in the MCM port configuration register (PCR) at CCSR offset 0x0_1010. If the DIU is operational (that is, DIU_MODE ≠ 000) and then it is turned off (DIU_MODE = 000), the user must wait at least 100 µs before re-enabling the DIU.

[1] Writing a reserved value is blocked and does not affect the register value.

10.3.1.9 BGND Register

The BGND register, shown in Figure 10-9, sets default background color for Plane 1. This is the color used to fill the areas in Plane 1 for which no data is assigned in the area descriptor.

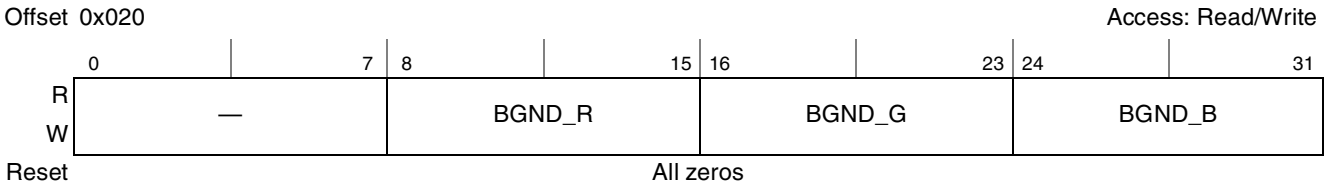


Figure 10-9. BGBD Register

See Table 10-11 for BGBD field descriptions.

Table 10-11. BGBD Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29–31	DIU_MODE	DIU Operation Mode 000 Mode 0: DIU off. 001 Mode 1: All three planes output to display. 010 Mode 2: Plane 1 to display, Planes 2 and 3 written back to memory. 011 Mode 3: All three planes written back to memory. 100 Mode 4: Color bar generation. All other encodings are reserved[1] **Note:** The DIU must not be enabled (DIU_MODE ≠ 000) if the core is in a boot hold off state. Software can determined the boot holdoff state of the core by polling the PORT0_EN bit in the MCM port configuration register (PCR) at CCSR offset 0x0_1010. If the DIU is operational (that is, DIU_MODE ≠ 000) and then it is turned off (DIU_MODE = 000), the user must wait at least 100 µs before re-enabling the DIU.

[1] Writing a reserved value is blocked and does not affect the register value.

NOTE

Programming the BGND register at the middle of a frame affects the display immediately, so one should reprogram it after a VSYNC or VSYNC_WB interrupt is detected.

10.3.1.10 BGND_WB Register

The BGND_WB register, shown in Figure 10-10, sets default background color for write back planes. This is the color used to fill the areas for which no data is assigned in the area descriptor.

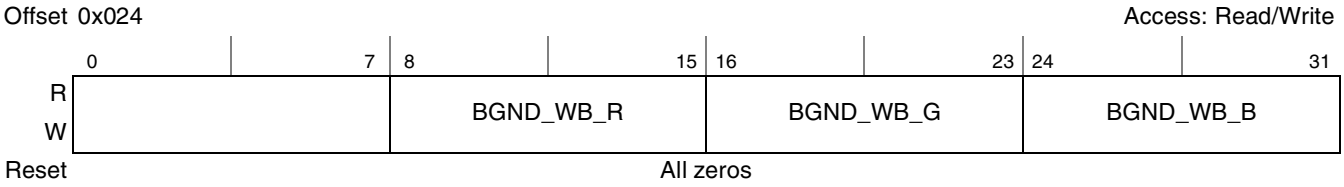


Figure 10-10. BGBD_WB Register

See Table 10-11 for BGBD field descriptions.

Table 10-12. BGBD Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	BGND_WB_R	BGND_WB_R represents the red component of the background for the write back planes. Note that bit 8 is the most significant bit and bit 15 is the least significant bit of the red component.

Table 10-12. BGBD Field Descriptions (continued)

Bits	Name	Description
16–23	BGND_WB_G	BGND_WB_G represents the green component of the background for the write back planes. Note that bit 16 is the most significant bit and bit 23 is the least significant bit of the green component.
24–31	BGND_WB_B	BGND_WB_B represents the blue component of the background for the write back planes. Note that bit 24 is the most significant bit and bit 31 is the least significant bit of the blue component.

NOTE

Programming the BGND_WB register at the middle of a frame affects the display immediately, so one should reprogram it after a VSYNC_WB or VSYNC interrupt is detected.

10.3.1.11 DISP_SIZE Register

Figure 10-11 shows the DISP_SIZE register.

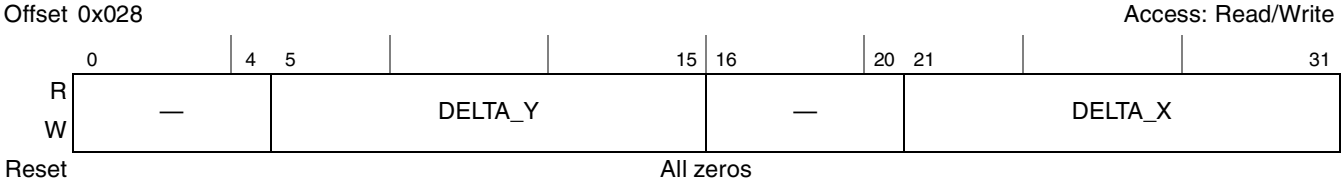


Figure 10-11. DISP_SIZE Register

Table 10-13 describes the DISP_SIZE fields.

Table 10-13. DISP_SIZE Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–15	DELTA_Y	DISP_SIZE register sets the display size (in pixels). Fields DELTA_X, DELTA_Y represents horizontal, vertical resolution separately.
16–20	—	Reserved
21–31	DELTA_X	DISP_SIZE register sets the display size (in pixels). Fields DELTA_X, DELTA_Y represents horizontal, vertical resolution separately.

10.3.1.12 WB_SIZE Register

The WB_SIZE register, shown in Figure 10-12, sets the write back frame size (in pixels).

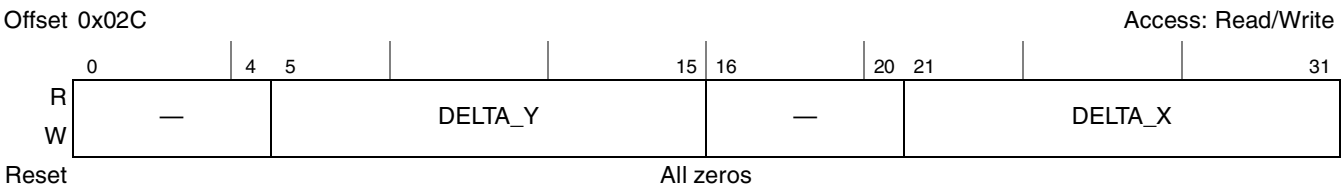


Figure 10-12. WB_SIZE Register

Table 10-14 describes the DISP_SIZE fields.

Table 10-14. WB_SIZE Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–15	DELTA_Y_WB	DELTA_Y_WB represents the vertical resolution.
16–20	—	Reserved
21–31	DELTA_X_WB	DELTA_X_WB represents the horizontal resolution.

10.3.1.13 WB_MEM_ADDR Register

Figure 10-13 shows the WB_MEM_ADDR register.

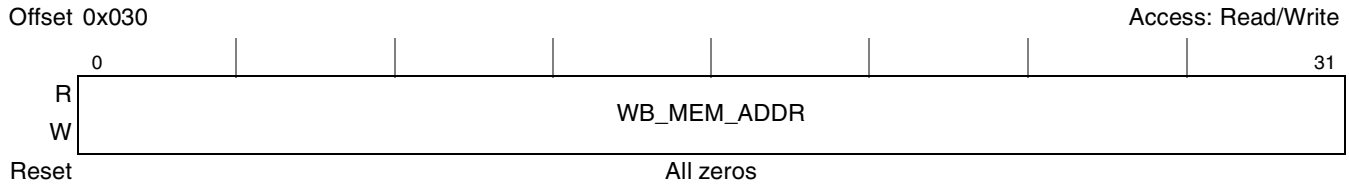


Figure 10-13. WB_MEM_ADDR Register

Table 10-15 describes the WB_MEM_ADDR fields.

Table 10-15. WB_MEM_ADDR Field Descriptions

Bits	Name	Description
0–31	WB_MEM_ADDR	WB_MEM_ADDR register sets the base address where the write back frame is written to in the memory. Write to this register will trigger a write back frame refresh. This address must be 64-bit boundary aligned (set the lowest 3 bits to 0); 32-byte boundary aligned address is more efficient.

10.3.1.14 HSYN_PARA Register

The HSYN_PARA register, shown in Figure 10-14, sets timing parameters related to the horizontal synchronization signal generation. Field FP_H, BP_H, PW_H stands for HSYNC signal front-porch, back-porch, and active pulse width separately. See Figure 10-50 for detailed signal meaning.

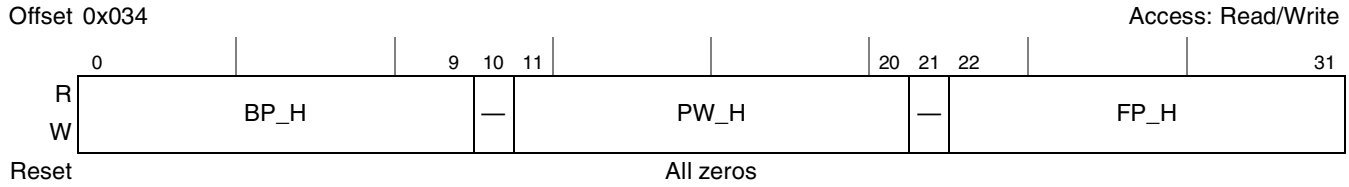


Figure 10-14. HSYN_PARA Register

Table 10-16 describes the HSYN_PARA fields.

Table 10-16. HSYN_PARA Field Descriptions

Bits	Name	Description
0–9	BP_H	HSYNC back-porch pulse width (in pixel clock cycles, it can be 0x0).
10	—	Reserved
11–20	PW_H	HSYNC active pulse width (in pixel clock cycles). PW_H must be greater than or equal to 1.
21	—	Reserved
22–31	FP_H	HSYNC front-porch pulse width (in pixel clock cycles, it can be 0x0).

10.3.1.15 VSYN_PARA Register

The VSYN_PARA register, shown in Figure 10-15, sets timing parameters related to the vertical synchronization signal generation. Field FP_V, BP_V, PW_V stands for VSYNC signal front-porch, back-porch, and active pulse width separately. See Figure 10-51, the display timing diagram for detailed signal meaning.

Offset 0x038 Access: Read/Write

	0	9	10	11	20	21	22	31
R W	BP_V		—	PW_V		—	FP_V	
Reset	All zeros							

Figure 10-15. VSYN_PARA Register

Table 10-17 describes the VSYN_PARA fields.

Table 10-17. VSYN_PARA Field Descriptions

Bits	Name	Description
0–9	BP_V	VSYNC back-porch pulse width (in HSYNC signal cycles, it can be 0x0).
10	—	Reserved
11–20	PW_V	VSYNC active pulse width (in HSYNC signal cycles). PW_V must be greater than or equal to 1.
21	—	Reserved
22–31	FP_V	VSYNC front-porch pulse width (in HSYNC signal cycles, it can be 0x0).

10.3.1.16 SYN_POL Register

The SYN_POL register, shown in Figure 10-16, selects polarity for corresponding synchronize signals (HSYNC, VSYNC, CSYNC), and control the bypass of HSYNC or VSYNC with CSYNC signal.

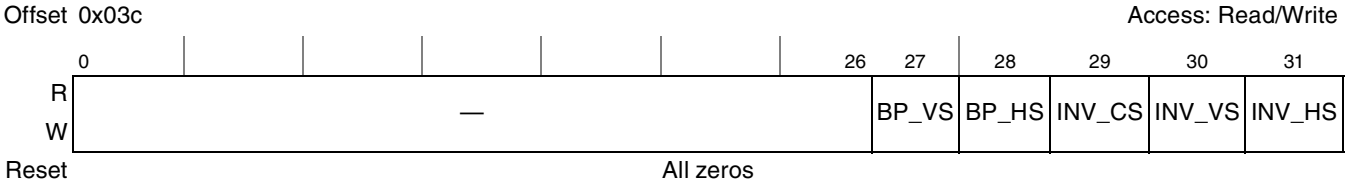


Figure 10-16. SYN_POL Register

Table 10-18 describes the SYN_POL fields.

Table 10-18. SYN_POL Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27	BP_VS	Bypass Vertical Synchronize Signal (internal pin muxing). 0 Not bypass VSYNC signal output 1 CSYNC bypass VSYNC signal, output CSYNC instead of VSYNC
28	BP_HS	Bypass Horizontal Synchronize Signal (internal pin muxing). 0 Not bypass HSYNC signal output 1 CSYNC bypass HSYNC signal, output CSYNC instead of HSYNC
29	INV_CS	Invert Composite Synchronize Signal. 0 Not invert CSYNC signal, active HIGH 1 Invert CSYNC signal, active LOW
30	INV_VS	Invert Vertical Synchronize Signal. 0 Not invert VSYNC signal, active HIGH 1 Invert VSYNC signal, active LOW
31	INV_HS	nvert Horizontal Synchronize Signal. 0 1 Not invert HSYNC signal, active HIGH 1 Invert HSYNC signal, active LOW

10.3.1.17 THRESHOLDS Register

The THRESHOLDS register, shown in Figure 10-17, sets three useful threshold values related to DIU operations.

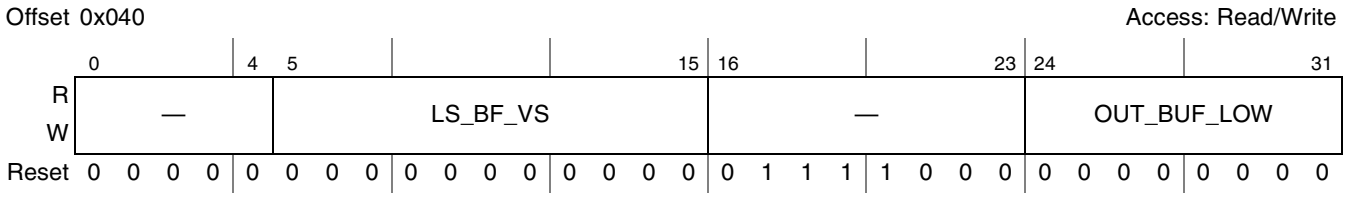


Figure 10-17. THRESHOLDS Register

Table 10-19 describes the THRESHOLDS fields.

Table 10-19. THRESHOLDS Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–15	LS_BF_VS	Lines before vsync threshold, This is a threshold value used to generate the ls_bf_vs interrupt status. Sets the number of lines ahead of vertical front porch (FP_V) when the interrupt will be generated.
16–23	—	Reserved
24–31	OUT_BUF_LOW	Output buffer filling low threshold (in pixels), This is used to generate the buffer under run exception. An under run exception is generated if display needs data and output buffer filling is lower than or equal to the OUT_BUF_LOW threshold.

10.3.1.18 INT_STATUS Register

The INT_STATUS register, shown in Figure 10-17, indicates the interrupt status. The DIU has only one interrupt signal. The CPU will read the INT_STATUS register to decide which exception occurs when an interrupt is detected. The read operation will also clear the register.

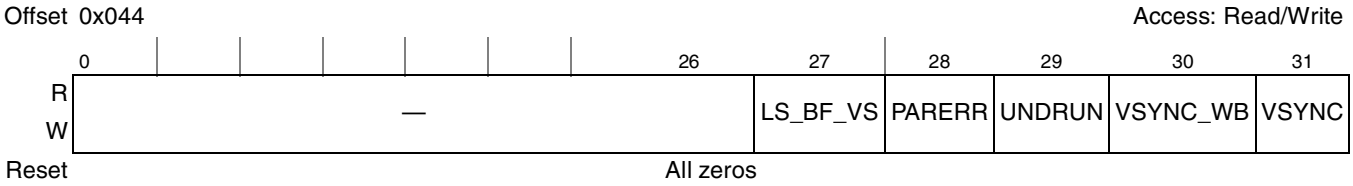


Figure 10-18. INT_STATUS Register

Table 10-20 describes the INT_STATUS fields.

Table 10-20. INT_STATUS Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27	LS_BF_VS	Lines before VSync interrupt is generated threshold “LS_BF_VS” number of lines ahead of the vertical front porch (FP_V) if enabled.
28	PAR	Display parameters error interrupt is generated if the user sets the display parameters wrongly.
29	UNDRUN	Under run exception interrupt is asserted when display needs data and output buffer filling is lower than or equal to the OUT_BUF_LOW threshold.
30	VSYNC_WB	Vertical synchronization interrupt for write back operation is an interrupt generated at the end of a write back frame. Used in mode2 and 3 only.
31	VSYNC	Vertical synchronization interrupt. If enabled, this interrupt will be generated at the beginning of a frame.

10.3.1.19 INT_MASK Register

Figure 10-19 shows the INT_MASK register.

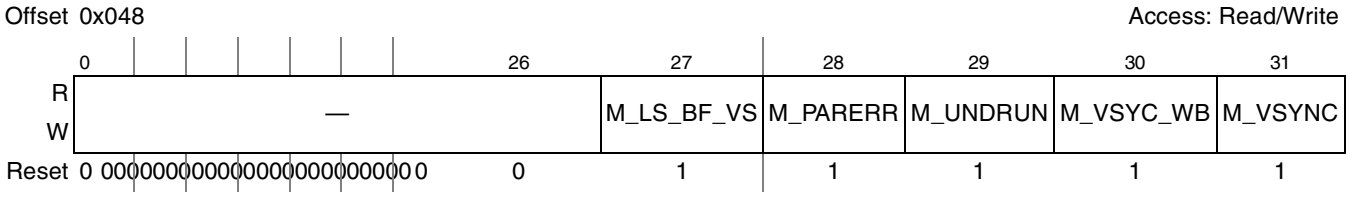


Figure 10-19. INT_MASK Register

Table 10-21 describes the INT_MASK fields.

Table 10-21. INT_MASK Field Descriptions

Bits	Name	Description
0–26	—	Reserved
27	M_LS_BF_VS	NT_MASK register enables or masks corresponding interrupt status to become an interrupt. 1 mask the interrupt 0 enable the interrupt
28	M_PARERR	NT_MASK register enables or masks corresponding interrupt status to become an interrupt. 1 mask the interrupt 0 enable the interrupt
29	M_UNDRUN	NT_MASK register enables or masks corresponding interrupt status to become an interrupt. 1 mask the interrupt 0 enable the interrupt
30	M_VSYC_WB	NT_MASK register enables or masks corresponding interrupt status to become an interrupt. 1 mask the interrupt 0 enable the interrupt
31	M_VSYNC	NT_MASK register enables or masks corresponding interrupt status to become an interrupt. 1 mask the interrupt 0 enable the interrupt

10.3.1.20 COLBAR Registers

The COLBAR registers are used to generate color bars in functional test mode. Eight different pixel values are taken as input data, to display 8 color bars on the display. After reset. they take default values, including 0xff000000 (Black), 0xff0000ff (Blue), 0xff007f7f (Cyan), 0xff00ff00 (Green), 0xff7f7f00 (Yellow), 0xffff0000 (Red), 0xff7f007f (Purple), 0xffffffff (White).

NOTE

Programming the COLBAR Registers at the middle of a frame affects the display immediately, so one should reprogram them after VSYNC interrupt is detected.

10.3.1.20.1 COLBAR_1 Register

Offset 0x04C Access: Read/Write

	0	7	8	15	16	23	24	31
R W	—		COLBAR_1_R		COLBAR_1_G		COLBAR_1_B	
Reset	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0

Figure 10-20. COLBAR_1 Register (Black)

10.3.1.20.2 COLBAR_2 Register

Offset 0x050 Access: Read/Write

	0	7	8	15	16	23	24	31
R W	—		COLBAR_2_R		COLBAR_2_G		COLBAR_2_B	
Reset	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1

Figure 10-21. COLBAR_2 Register (Blue)

10.3.1.20.3 COLBAR_3 Register

Offset 0x054 Access: Read/Write

	0	7	8	15	16	23	24	31
R W	—		COLBAR_3_R		COLBAR_3_G		COLBAR_3_B	
Reset	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1	0 1 1 1	1 1 1 1

Figure 10-22. COLBAR_3 Register (Cyan)

10.3.1.20.4 COLBAR_4 Register

Offset 0x058 Access: Read/Write

	0	7	8	15	16	23	24	31
R W	—		COLBAR_4_R		COLBAR_4_G		COLBAR_4_B	
Reset	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0

Figure 10-23. COLBAR_4 Register (Green)

10.3.1.20.5 COLBAR_5 Register

Offset 0x05C Access: Read/Write

	0	7	8	15	16	23	24	31
R W	—		COLBAR_5_R		COLBAR_5_G		COLBAR_5_B	
Reset	1 1 1 1	1 1 1 1	0 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	0 0 0 0	0 0 0 0

Figure 10-24. COLBAR_5 Register (Yellow)

10.3.1.20.6 COLBAR_6 Register

Offset 0x060 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	—	COLBAR_6_R	COLBAR_6_G	COLBAR_6_B
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 0 0

Figure 10-25. COLBAR_6 Register (Red)

10.3.1.20.7 COLBAR_7 Register

Offset 0x064 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	—	COLBAR_7_R	COLBAR_7_G	COLBAR_7_B
Reset	1 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0	0 1 1 1 1 1 1 1

Figure 10-26. COLBAR_7 Register (Purple)

10.3.1.20.8 COLBAR_8 Register

Offset 0x068 Access: Read/Write

	0–7	8–15	16–23	24–31
R W	—	COLBAR_8_R	COLBAR_8_G	COLBAR_8_B
Reset	All ones			

Figure 10-27. COLBAR_8 Register (White)

10.3.1.21 FILLING Register

FILLING register, shown in Figure 10-28, is a read-only register for debug purpose. It indicates current filling status of the input and output buffers.

Offset 0x06c Access: Read only

	0–7	8–15	16–19	20–23	24–27	28–31
R	—	FILLING_OBF	FILLING_WB	FILLING_P3	FILLING_P2	FILLING_P1
W						
Reset	All zeros					

Figure 10-28. FILLING Register

Table 10-22 describes the FILLING fields.

Table 10-22. FILLING Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	FILLING_OBF	Filling status of the output buffer (in pixels, 24 bits per pixel).

Table 10-22. FILLING Field Descriptions (continued)

Bits	Name	Description
16–19	FILLING_WB	Filling status of the write back pixel buffer (number of filled buffers out of the 8 256-byte buffers).
20–23	FILLING_P3	Filling status of Plane 3 input pixel buffer (number of filled buffers out of the 8 256-byte buffers).
24–27	FILLING_P2	Filling status of Plane 2 input pixel buffer (number of filled buffers out of the 8 256-byte buffers).
28–31	FILLING_P1	Filling status of Plane 1 input pixel buffer (number of filled buffers out of the 8 256-byte buffers).

10.3.1.22 PLUT Register

The PLUT register, shown in Figure 10-29, determines the priority of the DIU transactions relative to other initiators on the internal platform bus.

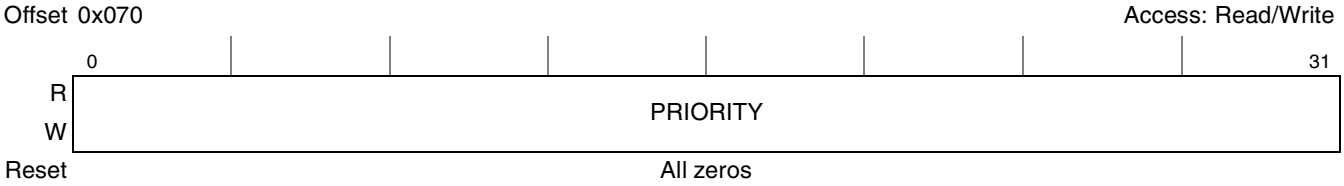


Figure 10-29. PLUT Register

Table 10-29 describes the PLUT fields.

Table 10-23. PLUT Field Descriptions

Bits	Name	Description
0–31	PRIORITY	Used to control the priority for the DIU. There are four recommended settings that can be used for any DIU applications to alter the relative bandwidth offered the DIU versus the core and other initiators. If a DIU underrun occurs, then a higher bandwidth DIU setting can be used. 0x0000_0000 Low bandwidth DIU, with maximum bandwidth available to other initiators. This setting is recommended when the display size is less than VGA (640×480) and only one plane is used. 0x000F_F5F6 Low-medium bandwidth DIU. This setting is recommended when the display size is greater than or equal to VGA (640×480) but less than XGA (1024×768) and only one plane is used. 0x0011_1F66 Medium-high bandwidth DIU with low (less than 0.5GByte/s combined) PCI/PCI-Express bandwidth. This setting is recommended when the display size is greater than or equal to XGA (1024×768) and one plane is used, or when the display size is less than XGA (1024×768) and multiple planes are used. 0x001F_5F66 Medium-high bandwidth DIU with high (greater than 0.5GByte/s combined) PCI/PCI-Express bandwidth. This setting is recommended when the display size is greater than or equal to XGA (1024×768) and one plane is used, or when the display size is less than XGA (1024×768) and multiple planes are used. 0x0111_F666 High bandwidth DIU with low (less than 0.5GByte/s combined) PCI/PCI-Express bandwidth. This setting is recommended when the display size is greater than or equal to XGA (1024×768) and multiple planes are used. 0x01F5_F666 High bandwidth DIU with high (greater than 0.5GByte/s combined) PCI/PCI-Express bandwidth. This setting is recommended when the display size is greater than or equal to XGA (1024×768) and multiple planes are used. Other values are reserved.

10.4 Functional Description

The DIU does not have internal frame buffers. It reads the data from main memory at the same rate at which it refreshes the display.

Besides generating all the signals required to drive the display, the DIU handles the real time blending of up to three planes onto the display. Alpha blending is performed between the planes. Chroma key support is also present to help relieve the host processor from all the very computational and bandwidth consuming blending tasks while simultaneously allowing the users to maintain the graphics quality required by many applications.

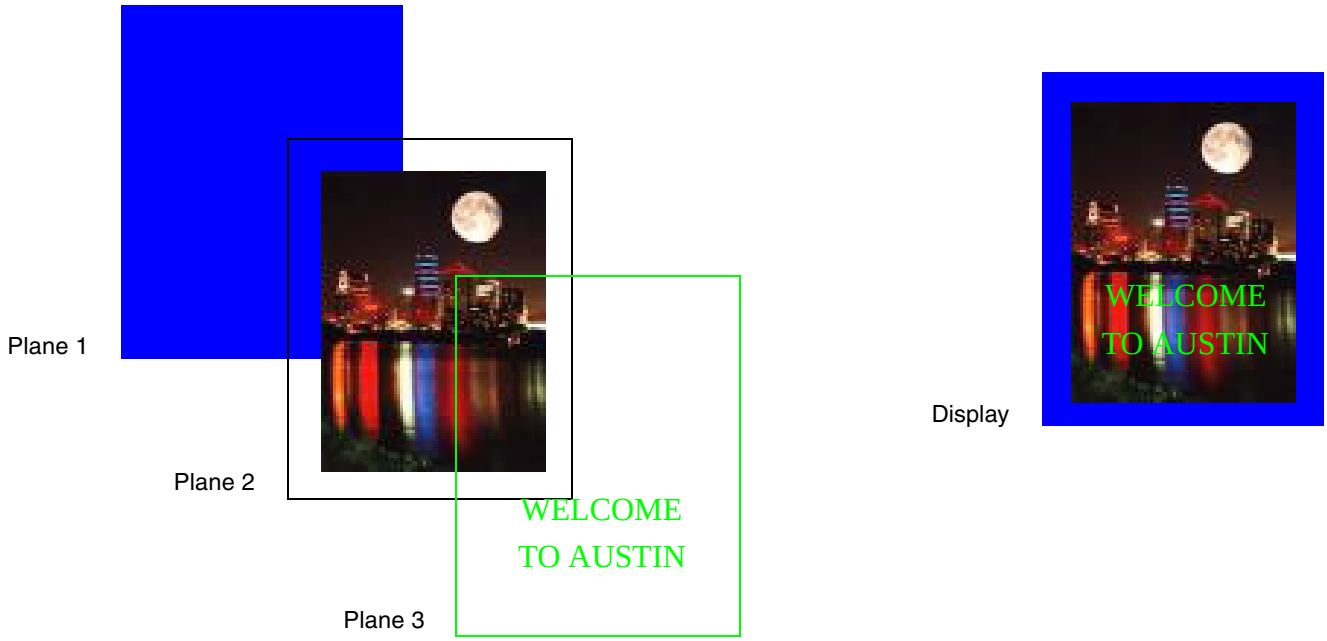


Figure 10-30. Plane Blending

10.4.1 The Area Descriptor

The area descriptor (AD) is the structure that defines each area to be displayed on a plane. A plane can display more than one area as long as they don’t share a scan line. Note the following restrictions on how many areas may be used by the DIU planes, depending on which mode is selected:

- Display planes, available in DIU modes 1 and 2, support a maximum of two areas (that is, a maximum of two area descriptors per display plane)
- Write back planes, available in DIU modes 2 and 3, support a maximum of one area (that is, a maximum of one area descriptor per write back plane)

The areas (for a plane) must be sorted in vertical order from the top to the bottom. The area descriptor is set up in the system’s main memory and then retrieved by the DIU directly from there.

NOTE

The DIU must use either the DDR or eLBC controllers for its memory; the MPC8610 does not support access to the PCI or PCI Express memory space by the DIU. For highest performance, the DIU should use the DDR memory space for its area descriptors and pixel data.

To change the displayed data between frames, the user can either change the data in the current area descriptor or create a new one and change the pointer in the DIU while keeping the previous one for future use or reference. It is always assumed that the bitmaps are stored pixel by pixel in memory, starting from the top-left most pixel in the image and continued sequentially until the last pixel (the bottom, right most pixel) by scanning the image always from left to right and top to bottom.

Figure 10-31 and Figure 10-32 show graphical representations of the area descriptor parameters that define an area. Figure 10-31 shows the parameters that specify how the source bitmap located in memory is interpreted by the DIU. The user might want to display only a limited area of the bitmap, by specifying an area of interest (AOI) as a subset of this bitmap.

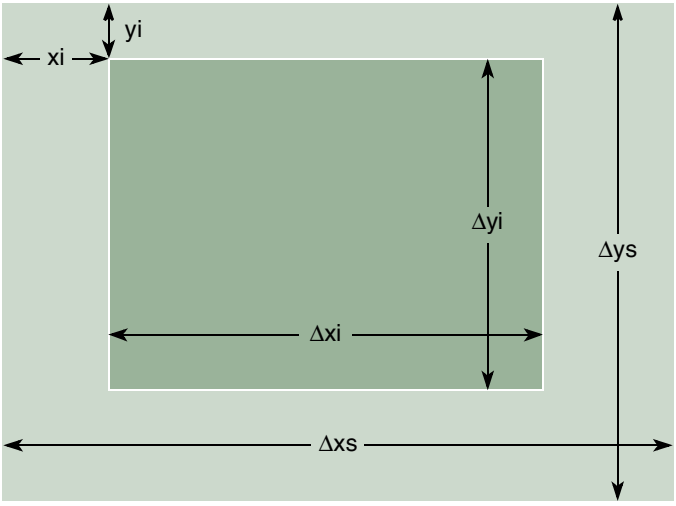


Figure 10-31. Source Bitmap Parameters

The complete source bitmap is specified by its starting address in memory, its pixel format and its dimensions. The subset to be displayed is defined by its relative position to the beginning of the bitmap (top left corner) and its own size which can be as large as the original image for the case where the whole image is to be displayed. The DIU automatically fetches the data, optimizing the accesses to minimize the bandwidth consumed by the operation.

There are some limitations on the size of an AOI, which include:

- the height of the AOI (Δyi) must be greater than or equal to 4 pixels
- the pixel data per line (Δxi × pixel size) must be greater than or equal to 8 bytes.

Note that using a very small AOI, such as one significantly smaller than 256 bytes, could cause the DIU to be more sensitive to DDR access latency. Therefore, it is recommended not to use very small AOIs. It is also recommended to leave a few lines in between AOIs in a display plane whenever possible for better tolerance of DDR access latency.

Figure 10-32 shows the parameters that specify how the image is to be displayed.

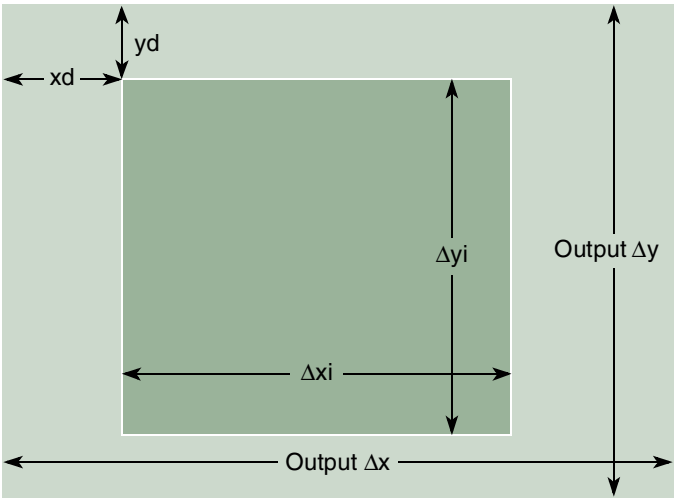


Figure 10-32. Display Parameters

There are two parameters that determine how is this image to be displayed, the first is its relative position with respect to the top-left corner of the display and the second is its orientation (by flipping the image in its x-axis or y-axis, but not both simultaneously). Notice that the size of the display is not part of the area descriptor as it is common to all area descriptors of all planes.

10.4.1.1 Area Descriptor Format

Each AD is composed of a ten-word data structure. The general format for an AD is shown in Figure 10-33.

Offset	
0x00	Word 0—Pixel format
0x04	Word 1—Bitmap address
0x08	Word 2—Source size/Global alpha
0x0C	Word 3—AOI size
0x10	Word 4—AOI offset
0x14	Word 5—Display offset
0x18	Word 6—Chroma key max
0x1C	Word 7—Chroma key min
0x20	Word 8—Next AD
0x24	Word 9—Reserved

Figure 10-33. Area Descriptor Format

The following sections describe the individual components that make up the area descriptor.

NOTE

The area descriptor uses little-endian byte ordering. A 32-bit word endian swap must be done for each word before writing it to memory.

10.4.1.1.1 Area Descriptor Word 0—Pixel Format

Figure 10-34 shows the fields of AD Word 0, which defines the pixel format.

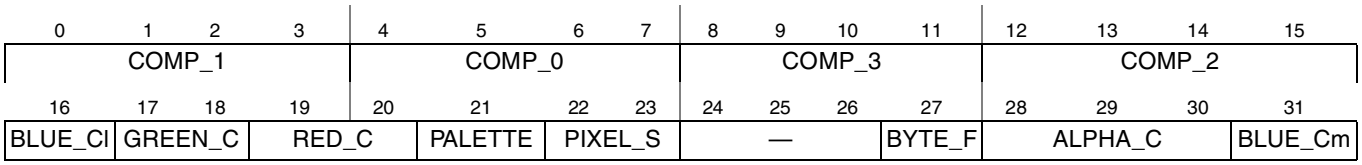


Figure 10-34. Area Descriptor Word 0—Pixel Format

Table 10-24 describes the pixel format fields.

Table 10-24. Area Descriptor Word 0—Pixel Format

Bits	Name	Description
0–3	COMP_1	Number of bits for component 1. Valid range is 0 through 8.
4–7	COMP_0	Number of bits for component 0. Valid range is 0 through 8.
8–11	COMP_3	Number of bits for component 3. Valid range is 0 through 8.
12–15	COMP_2	Number of bits for component 2. Valid range is 0 through 8.
16	BLUE_CI	Blue component assignment (lsb). This bit, together with BLUE_Cm, assigns the component number used for the blue channel. 00 Component 0 01 Component 1 10 Component 2 11 Component 3 Default value is 00.
17–18	GREEN_C	Green component assignment. This field assigns the component number used for the green channel. 00 Component 0 01 Component 1 10 Component 2 11 Component 3 Default value is 01.
19–20	RED_C	Red component assignment. This field assigns the component number used for the red channel. 00 Component 0 01 Component 1 10 Component 2 11 Component 3 Default value is 10.
21	PALETTE	Palette mode. Determines whether palette mode is used. 0 Palette mode disabled 1 Palette mode enabled

Table 10-24. Area Descriptor Word 0—Pixel Format (continued)

Bits	Name	Description
22–23	PIXEL_S	Pixel size. Specifies the number bytes per pixel. Note the actual number of bytes per pixel is PIXEL_S+1 00 1 byte per pixel 01 2 bytes per pixel 10 3 bytes per pixel 11 4 bytes per pixel
24–26	—	Reserved
27	BYTE_F	Byte flip disable. When cleared, flips the byte order for the pixel data. See Section 10.4.2, "Pixel Structure," for more information. 0 Bytes are flipped 1 Bytes are not flipped (recommended)
28–30	ALPHA_C	Alpha component assignment. This field assigns the component number used for the alpha channel. 000 Component 0 001 Component 1 010 Component 2 011 Component 3 100 Global alpha Default value is 011.
31	BLUE_Cm	Blue component assignment (msb). Together with BLUE_Cl assigns the component number used for the blue channel. 00 Component 0 01 Component 1 10 Component 2 11 Component 3 Default value is 00.

10.4.1.1.2 Area Descriptor Word 1—Bitmap Address

Figure 10-35 shows the fields of AD Word 1, which defines the bitmap address.

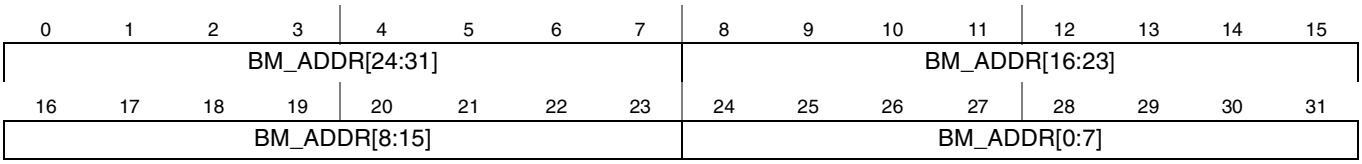


Figure 10-35. Area Descriptor Word 1—Bitmap Address

Table 10-25 describes the bitmap address fields.

Table 10-25. Area Descriptor Word 1—Bitmap Address

Bits	Name	Description
0–7	BM_ADDR[24:31]	Bits 24 through 31 (LSB) of the bitmap address pointer
8–15	BM_ADDR[16:23]	Bits 16 through 23 of the bitmap address pointer
16–23	BM_ADDR[8:15]	Bits 8 through 15 of the bitmap address pointer
23–31	BM_ADDR[0:7]	Bits 0 through 7 (MSB) of the bitmap address pointer

10.4.1.1.3 Area Descriptor Word 2—Source Size/Global Alpha

Figure 10-36 shows the fields of AD Word 2, which defines the source size and the global alpha value.

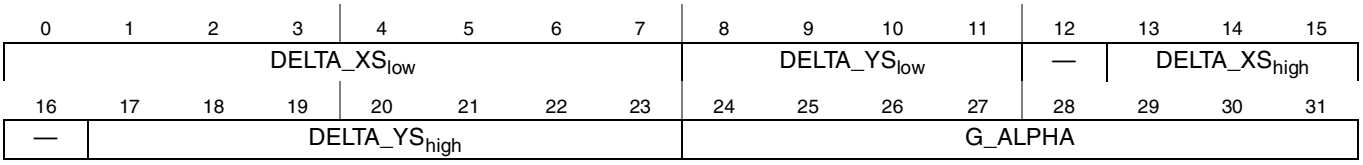


Figure 10-36. Area Descriptor Word 2—Source Size/Global Alpha

Table 10-26 describes the source size/global alpha fields.

Table 10-26. Area Descriptor Word 2—Source Size/Global Alpha

Bits	Name	Description
0–7	$DELTA\_XS_{low}$	Lower order bits of the Δxsi parameter that defines the horizontal size (in pixels) of the source bitmap.
8–11	$DELTA\_YS_{low}$	Lower order bits of the Δys parameter that defines the vertical size (in pixels) of the source bitmap.
12	—	Reserved
13–15	$DELTA\_XS_{high}$	Higher order bits of the Δxs parameter that defines the horizontal size (in pixels) of the source bitmap.
16	—	Reserved
17–23	$DELTA\_YS_{high}$	Higher order bits of the Δys parameter that defines the vertical size (in pixels) of the source bitmap.
24–31	G_ALPHA	Global alpha. Value used for the alpha channel for all pixels in the area when the data format does not include an alpha component or when the user wants to override the alpha values in the data (ALPHA_C = 100). Note that pixels to be displayed on Plane 1 ignore the alpha value.

10.4.1.1.4 Area Descriptor Word 3—AOI Size

Figure 10-37 shows the fields of AD Word 3, which defines the size of the area of interest.

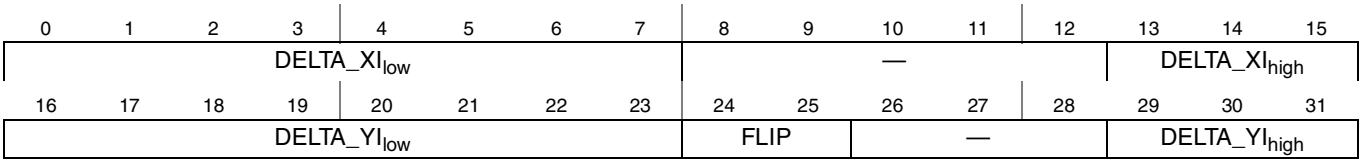


Figure 10-37. Area Descriptor Word 3—AOI Size

Table 10-27 describes the area of interest size fields.

Table 10-27. Area Descriptor Word 3—AOI Size

Bits	Name	Description
0–7	$DELTA\_XI_{low}$	Lower order bits of the Δxi parameter that defines the horizontal size (in pixels) of the area of interest.
8–12	—	Reserved
13–15	$DELTA\_XI_{high}$	Higher order bits of the Δxi parameter that defines the horizontal size (in pixels) of the area of interest.

Table 10-27. Area Descriptor Word 3—AOI Size (continued)

Bits	Name	Description
16–23	$DELTA\_YI_{low}$	Lower order bits of the Δyi parameter that defines the vertical size (in pixels) of the area of interest.
24–25	FLIP	Area of interest image flip. Flips the bitmap image either horizontally or vertically within the area of interest. 00 Normal 01 Flip image horizontally (about the y-axis) 10 Flip image vertically (about the x-axis) 11 Reserved
26–28	—	Reserved
29–31	$DELTA\_YI_{high}$	Higher order bits of the Δyi parameter that defines the vertical size (in pixels) of the area of interest.

10.4.1.1.5 Area Descriptor Word 4—AOI Offset

Figure 10-38 shows the fields of AD Word 4, which defines the offset for the area of interest.

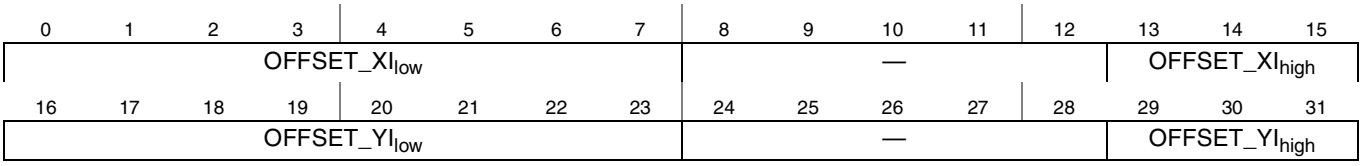


Figure 10-38. Area Descriptor Word 4—AOI Offset

Table 10-28 describes the area of interest offset fields.

Table 10-28. Area Descriptor Word 4—AOI Offset

Bits	Name	Description
0–7	$OFFSET\_XI_{low}$	Lower order bits of the xi parameter that defines the horizontal offset (in pixels) of the area of interest from the start of the source bitmap in memory.
8–12	—	Reserved
13–15	$OFFSET\_XI_{high}$	Higher order bits of the xi parameter that defines the horizontal offset (in pixels) of the area of interest from the start of the source bitmap in memory.
16–23	$OFFSET\_YI_{low}$	Lower order bits of the yi parameter that defines the vertical offset (in pixels) of the area of interest from the start of the source bitmap in memory.
24–28	—	Reserved
29–31	$OFFSET\_YI_{high}$	Higher order bits of the yi parameter that defines the vertical offset (in pixels) of the area of interest from the start of the source bitmap in memory.

10.4.1.1.6 Area Descriptor Word 5—Display Offset

Figure 10-39 shows the fields of AD Word 5, which defines the offset for the area of interest in the display.

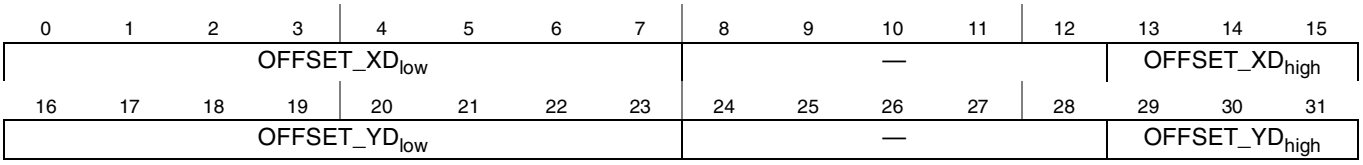


Figure 10-39. Area Descriptor Word 5—Display Offset

Table 10-29 describes the display offset fields.

Table 10-29. Area Descriptor Word 5—Display Offset

Bits	Name	Description
0–7	$OFFSET\_XD_{low}$	Lower order bits of the xd parameter that defines the horizontal offset (in pixels) of the area of interest in the display.
8–12	—	Reserved
13–15	$OFFSET\_XD_{high}$	Higher order bits of the xd parameter that defines the horizontal offset (in pixels) of the area of interest in the display.
16–23	$OFFSET\_YD_{low}$	Lower order bits of the yd parameter that defines the vertical offset (in pixels) of the area of interest in the display.
24–28	—	Reserved
29–31	$OFFSET\_YD_{high}$	Higher order bits of the yd parameter that defines the vertical offset (in pixels) of the area of interest in the display.

10.4.1.1.7 Area Descriptor Word 6—Chroma Key Max

Figure 10-40 shows the fields of AD Word 6, which defines the maximum values for chroma key. See Section 10.4.2.4, “Chroma Keying,” for more information.

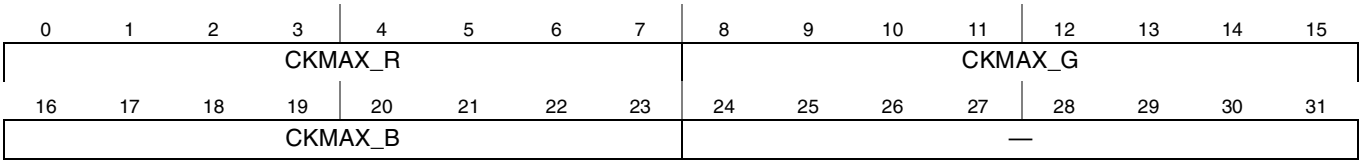


Figure 10-40. Area Descriptor Word 6—Chroma Key Max

Table 10-30 describes the chroma key min fields.

Table 10-30. Area Descriptor Word 6—Chroma Key Max

Bits	Name	Description
0–7	CKMAX_R	Chroma key maximum red component value
8–15	CKMAX_G	Chroma key maximum green component value
16–23	CKMAX_B	Chroma key maximum blue component value
23–31	—	Reserved

10.4.1.1.8 Area Descriptor Word 7—Chroma Key Min

Figure 10-41 shows the fields of AD Word 7, which defines the minimum values for chroma key. See Section 10.4.2.4, "Chroma Keying," for more information.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CKMIN_R								CKMIN_G							
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CKMIN_B								—							

Figure 10-41. Area Descriptor Word 7—Chroma Key Min

Table 10-31 describes the chroma key min fields.

Table 10-31. Area Descriptor Word 7—Chroma Key Min

Bits	Name	Description
0–7	CKMIN_R	Chroma key minimum red component value
8–15	CKMIN_G	Chroma key minimum green component value
16–23	CKMIN_B	Chroma key minimum blue component value
23–31	—	Reserved

10.4.1.1.9 Area Descriptor Word 8—Next AD

Figure 10-42 shows the fields of AD Word 8, which defines the next AD address. If more than one area is to be displayed in the same plane, the next AD points to the next area descriptor. If this is the last AD in the current frame, the next AD address must be set to all zeros. The next AD address must be aligned to a 8-byte boundary (that is, the lowest three bits should be 000).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
NEXT_AD_ADDR[24:31]								NEXT_AD_ADDR[16:23]							
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
NEXT_AD_ADDR[8:15]								NEXT_AD_ADDR[0:7]							

Figure 10-42. Area Descriptor Word 8—Next AD

Table 10-32 describes the next area descriptor address fields.

Table 10-32. Area Descriptor Word 8—Next AD

Bits	Name	Description
0–7	NEXT_AD_ADDR[24:31]	Bits 24 through 31 (LSB) of the next area descriptor address pointer
8–15	NEXT_AD_ADDR[16:23]	Bits 16 through 23 of the next area descriptor address pointer
16–23	NEXT_AD_ADDR[8:15]	Bits 8 through 15 of the next area descriptor address pointer
23–31	NEXT_AD_ADDR[0:7]	Bits 0 through 7 (MSB) of the next area descriptor address pointer

NOTE

Display frames may have a maximum of two ADs, so only the first AD can have a non-zero NEXT_AD field. Also note that write back frames can only support a single AD, so the NEXT_AD field must be zero for write back mode ADs.

10.4.2 Pixel Structure

The DIU has been designed for maximum flexibility in the format of its input data. Each pixel can contain up to four basic components: alpha, red, green, and blue. Several parameters in an area descriptor (see Figure 10-34) determine how the DIU parses the bitmap data into the pixel components.

The first step is to define the size of each pixel (PIXEL_S+1) and the number of bits for each component of a pixel (COMP_0 to COMP_3) in the data stream. The next step is to assign the components of the pixel (alpha, R, G, and B) to the components of the data stream (COMP_0 to COMP_3). The RED_C, GREEN_C, BLUE_C, and ALPHA_C fields in the pixel format (word 0) of the AD are used to control the assignments, selecting which component is red, which is green and so on. The three color components (red, green, and blue) must be mapped to one of the components of the data stream; alpha can be assigned to a fourth component of the data stream or the pixel can use the global alpha value specified for the current area descriptor (G_ALPHA in word 2).

The pixel format (word 0) of the AD also contains the BYTE_F bit, which can have a significant impact on the way the DIU interprets the pixel data. If the BYTE_F bit in the AD is set (that is, flipping is disabled) then the pixel is constructed as shown in Figure 10-43.

Byte 0								Byte 1								Byte 2							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
COMP_3 (8-bits)								COMP_2 (5-bits)					COMP_1 (6-bits)			COMP_0 (5-bits)							
Alpha								Red					Green			Blue							

Figure 10-43. 24-bit ARGB 8:5:6:5 Pixel Structure Definition when BYTE_F = 1

If the BYTE_F bit in the AD is cleared (that is, bytes in the pixel are flipped) then the pixel is constructed as shown in Figure 10-44. The DIU performs the byte flipping on each pixel before it performs the mapping.

Byte 0								Byte 1								Byte 2							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
COMP_1 (3 lsbs)			COMP_0 (5-bits)					COMP_2 (5-bits)					COMP_1 (3 msbs)			COMP_3 (8-bits)							
Green_low			Blue					Red					Green_high			Alpha							

Figure 10-44. 24-bit ARGB 8:5:6:5 Pixel Structure Definition when BYTE_F = 0

10.4.2.1 Pixel Format Conversion

The DIU is designed to support a variety of pixel bit formats. Table 10-33 lists some examples of supported pixel formats.

Table 10-33. Examples of DIU supported pixel formats

Component Order 3, 2, 1, 0	Number of Bits per Component	Global Alpha	Palette
A:R:G:B	8:8:8:8	No	No
R:G:B	8:8:8	Yes	No
A:R:G:B	8:5:6:5	No	No
R:G:B	5:6:5	Yes	No
A:R:G:B	4:4:4:4	No	No
A:R:G:B	1:5:5:5	No	No
n/a	n/a	No	Yes
R:G:B:A	8:8:8:8	No	No
R:G:B:A	5:6:5:8	No	No
R:G:B:A	4:4:4:4	No	No
R:G:B:A	5:5:5:1	No	No
n/a	8	Yes	No

Internally all the calculations are performed using pixels represented by 8 bits per component. For those pixel formats in which the user specifies less than 8 bits per color component, the specified bits are used as the most significant bits and the lower significant bits are filled with zeros. The alpha values are extended to 8 bits too, but the lower significant bits are filled by sign extension to make sure that minimum value is 0 and the maximum value is extended to 255.

If the original data bitmap does not include alpha, the user can specify it using the global alpha field (G_ALPHA) in the area descriptor and/or use chroma keying.

10.4.2.2 Palette Mode

As an alternate to the larger bitmap formats, the DIU supports an 8-bit pixel format through a palette table (256 × 32-bit look up table). The palette table maps an 8-bit input pixel to a 32-bit color format. The palette table is stored in a 256 × 32 bit embedded SRAM. It is accessed by the hardware as an indexed matrix such that palettized_pixel [0–31] = palette_table[input_pixel [0–7]]. Note that palette mode is only supported for display frames; it is not supported for frames used for write back mode.

NOTE: Using Palette Mode on Plane 3 when in DIU Mode 1

When the DIU is in mode 1, and palette mode is used in plane 3 there is a possibility that a maximum of 1 pixel per AOI can be incorrect. When this occurs, the incorrect pixel contains the same value as the pixel located directly to its right. If this single pixel error per AOI is unacceptable, then palette mode should not be used in plane 3.

The palette table should be created in advance in memory as a consecutive sequence of 256 × 64 bits (with bits 32 to 63 filled with zeros). Writing to the PALETTE register causes the DIU to automatically reload a new palette table from memory before it starts processing the next frame. Figure 10-45 shows the format of the palette table in main memory.

Palette Entry	PALETTE Offset	Local Address [33:35]							
		000	001	010	011	100	101	110	111
0	0x000	P0_B	P0_G	P0_R	P0_A	0x00	0x00	0x00	0x00
1	0x008	P1_B	P1_G	P1_R	P1_A	0x00	0x00	0x00	0x00
2	0x010	P2_B	P2_G	P2_R	P2_A	0x00	0x00	0x00	0x00
3	0x018	P3_B	P3_G	P3_R	P3_A	0x00	0x00	0x00	0x00
. .									
254	0x7F0	P254_B	P254_G	P254_R	P254_A	0x00	0x00	0x00	0x00
255	0x7F8	P255_B	P255_G	P255_R	P255_A	0x00	0x00	0x00	0x00

Figure 10-45. Palette Table Format in Memory

10.4.2.3 Alpha Blending

For each pixel, besides the color components (R, G, B) there is a fourth component that defines the transparency of the pixel. This transparency value is called alpha and has a range between 0 (pixel is completely transparent) to 255 (pixel is completely opaque). The following equation represents the transfer function of the blending operation including the alpha values.

$$\text{out\_pixel} = \frac{\text{pixel\_1} \times (255 - \alpha_2) \times (255 - \alpha_3) + \text{pixel\_2} \times \alpha_2 \times (255 - \alpha_3) + \text{pixel\_3} \times 255 \times \alpha_3}{255^2}$$

Notice that there is no alpha component defined for plane1 since there are no planes behind it.

For mode 2, when only Plane 2 and 3 are blended to write back, the equation changes to the following:

$$\text{writeback} = \frac{\text{plane2} \times (255 - \alpha_3) + \text{plane3} \times \alpha_3}{255}$$

10.4.2.4 Chroma Keying

For each area (see Section 10.4.1, “The Area Descriptor,”) the user specifies a maximum and a minimum value for the chroma keying function. The maximum and minimum values are specified as R,G, and B values to which every pixel in the bitmap is compared and if all the components are greater or equal to the minimum and less than or equal to the max then the alpha component for that particular pixel is replaced with 0. The chroma keying operation is performed after the color format conversion with all the components extended to 8 bits.

To turn the chroma keying off simply set the minimum to 255 and the max to 0, for each component. If for example to produce a green-screen type of effect in which all green pixels (0,255,0) are to be turned transparent, the max and min should both be set to (0,255,0).

10.4.3 Gamma Correction

The gamma table allows the user to define a completely arbitrary transfer function at the output of each color component. The gamma table should be created in memory as a consecutive sequence of 256 bytes for each color component that will be accessed by the hardware as an indexed matrix such that output_color_component = gamma_table[input_color_component].

All three gamma tables must be stored in memory consecutively beginning with Red, then Green, and Blue at the end.

Similar to the palette, the gamma table is automatically loaded by the DIU from memory into an embedded SRAM before it starts on processing the next frame if the CPU writes to the GAMMA register. The size of the SRAM is 3 × 256 bytes, but it is organized as 96 × 64 bits. The address range for each color component table is:

- Gamma_red:0x000–0x0ff
- Gamma_green:0x100–0x1ff
- Gamma_blue:0x200–0x2ff.

Figure 10-46 shows the format of the gamma table in main memory.

Color Component	GAMMA Offset	Local Address [33:35]							
		000	001	010	011	100	101	110	111
Red	0x000	$G0_{red}$	$G1_{red}$	$G2_{red}$	$G3_{red}$	$G4_{red}$	$G5_{red}$	$G6_{red}$	$G7_{red}$
	0x008	$G8_{red}$	$G9_{red}$	$G10_{red}$	$G11_{red}$	$G12_{red}$	$G13_{red}$	$G14_{red}$	$G15_{red}$
		. . .							
	0x0F0	$G240_{red}$	$G241_{red}$	$G242_{red}$	$G243_{red}$	$G244_{red}$	$G245_{red}$	$G246_{red}$	$G247_{red}$
	0x0F8	$G248_{red}$	$G249_{red}$	$G250_{red}$	$G251_{red}$	$G252_{red}$	$G253_{red}$	$G254_{red}$	$G255_{red}$
Green	0x100	$G0_{green}$	$G1_{green}$	$G2_{green}$	$G3_{green}$	$G4_{green}$	$G5_{green}$	$G6_{green}$	$G7_{green}$
	0x108	$G8_{green}$	$G9_{green}$	$G10_{green}$	$G11_{green}$	$G12_{green}$	$G13_{green}$	$G14_{green}$	$G15_{green}$
		. . .							
	0x1F0	$G240_{green}$	$G241_{green}$	$G242_{green}$	$G243_{green}$	$G244_{green}$	$G245_{green}$	$G246_{green}$	$G247_{green}$
	0x1F8	$G248_{green}$	$G249_{green}$	$G250_{green}$	$G251_{green}$	$G252_{green}$	$G253_{green}$	$G254_{green}$	$G255_{green}$
Blue	0x200	$G0_{blue}$	$G1_{blue}$	$G2_{blue}$	$G3_{blue}$	$G4_{blue}$	$G5_{blue}$	$G6_{blue}$	$G7_{blue}$
	0x208	$G8_{blue}$	$G9_{blue}$	$G10_{blue}$	$G11_{blue}$	$G12_{blue}$	$G13_{blue}$	$G14_{blue}$	$G15_{blue}$
		. . .							
	0x2F0	$G240_{blue}$	$G241_{blue}$	$G242_{blue}$	$G243_{blue}$	$G244_{blue}$	$G245_{blue}$	$G246_{blue}$	$G247_{blue}$
	0x2F8	$G248_{blue}$	$G249_{blue}$	$G250_{blue}$	$G251_{blue}$	$G252_{blue}$	$G253_{blue}$	$G254_{blue}$	$G255_{blue}$

Figure 10-46. Gamma Table Format in Memory

10.4.4 Cursor

The DIU supports a 32 × 32 hardware cursor that is overlapped on top of all three planes. Figure 10-47 shows the structure of the cursor bitmap in memory. In Figure 10-47, each cursor pixel is labeled as C(*n*,*m*) where *n* indicates the row and *m* indicates the column of the 32 × 32 matrix. The cursor bitmap data is stored in memory as a continuous sequence of 16-bit pixels, starting from the top left corner, C(0,0), and continuing across the row (left to right) and then on to the next column (top to bottom) until the last cursor pixel, C(31,31), is at the bottom right corner.

Cursor Pixel Row	CURSOR Offset	Local Address [33:35]							
		000	001	010	011	100	101	110	111
0	0x000	C(0,0)		C(0,1)		C(0,2)		C(0,4)	
	0x008	C(0,5)		C(0,6)		C(0,7)		C(0,8)	
	. . .								
	0x038	C(0,28)		C(0,29)		C(0,30)		C(0,31)	
1	0x040	C(1,0)		C(1,1)		C(1,2)		C(1,4)	
	0x048	C(1,5)		C(1,6)		C(1,7)		C(1,8)	
	. . .								
	0x078	C(1,28)		C(1,29)		C(1,30)		C(1,31)	
2	0x080	C(2,0)		C(2,1)		C(2,2)		C(2,4)	
	0x088	C(2,5)		C(2,6)		C(2,7)		C(2,8)	
	. . .								
	0x0B8	C(2,28)		C(2,29)		C(2,30)		C(2,31)	
. . .									
31	0x7C0	C(31,0)		C(31,1)		C(31,2)		C(31,4)	
	. . .								
	0x7F0	C(31,24)		C(31,25)		C(31,26)		C(31,27)	
	0x7F8	C(31,28)		C(31,29)		C(31,30)		C(31,31)	

Figure 10-47. Cursor Structure in Memory

Each cursor pixel is in 16-bit, $G_L BARG_H$ 3:5:1:5:2 format, as shown in .

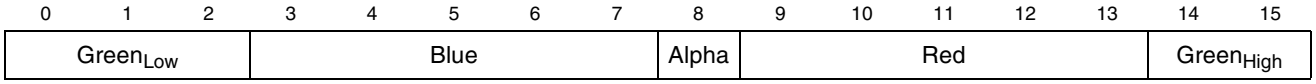


Figure 10-48. Cursor Pixel Format

The field descriptions for a cursor pixel are provided in Table 10-34.

Table 10-34. Field Descriptions for Cursor Pixel

Bit	Field	Description
0–2	Green$_{Low}$	Low order bits of the green component of cursor pixel C(*n*,*m*). The green component of the cursor pixel is made up of Green$_{High}$||Green$_{Low}$.
3–7	Blue	Blue component of cursor pixel C(*n*,*m*).
8	Alpha	Alpha component of cursor pixel C(*n*,*m*).

Table 10-34. Field Descriptions for Cursor Pixel

Bit	Field	Description
9–13	Red	Red component of cursor pixel C(*n*,*m*).
14–15	$Green_{High}$	High order bits of the green component of cursor pixel C(*n*,*m*). The green component of the cursor pixel is made up of $Green_{High}$||$Green_{Low}$.

To achieve cursors with shapes other than the basic square set the alpha value to 0 in those pixels that are to be transparent. To make the cursor disappear set the alpha value to 0 for all the pixels.

Similar to the palette and gamma, the cursor bitmap will also be automatically loaded by the DIU from memory into an embedded SRAM before it starts on processing the next frame if the CPU write to the CURSOR register (See Section 10.3.1.6, “CURSOR Register”). The SRAM size is, 256 × 64 bits.

The cursor position is determined by the register CURSOR_POS (See Section 10.3.1.7, “CURS_POS Register”); the specified coordinate corresponds to the top, left corner of the cursor bitmap. The cursor can be partially or totally off the screen. Note that programming the CURSOR_POS register in the middle of a frame affects the current cursor position immediately, so it is better to reprogram it after the VSYNC interrupt is detected.

10.4.5 Write-Back Operation

Besides driving the LCD display, the DIU supports two operating modes (modes 2 and 3) with memory write-back operation. Intermediate blending results are stored back in to the memory; these intermediate results can be then forwarded to other processing or display units, or be blended with other image source(s) again in the DIU, virtually extending the number of graphics planes.DIU Mode 1 supports the blending and display of up to 3 planes. If an application requires the display of an image blended from four or more planes, this can be accomplished in DIU Mode 2. Plane 1 is used to display one frame while the following frame is blended by the DIU. Planes 2 and 3 are used initially to read in the first and second planes to be combined, and the DIU writes back a combined plane 1+2. Then, while the DIU is still displaying the same frame on plane 1, the DIU reads back in on plane 2 the intermediate plane 1+2, and on plane 3 the new third plane, writing back a combined plane 1+2+3. Then (while the DIU is still displaying the same frame on plane 1), the DIU reads back in on plane 2 the intermediate plane 1+2+3, and on plane 3 the new fourth plane, writing back the combined plane 1+2+3+4. This combined plane is then ready to be displayed on plane 1 in the following frame, or extra steps can be added to blend more than four planes.

One caveat with this blending is the system bandwidth that is required. In the example above in which four planes are blended, for each frame displayed, the DIU must read in seven planes (the four original planes to be blended, as well as two intermediate planes and the final plane to be displayed), and write back three planes. This is a total of ten planes to be read or written by the DIU for each frame displayed. For example, if all planes have size XGA (1024×768) at 24-bits per pixel with 60 frames per second, the total bandwidth required by the DIU is 1.42 GByte/s. While this depends upon the individual application, it may not leave sufficient bandwidth for the core or other peripherals accessing DDR memory. Therefore, it is recommended that write-back for combined blending of more than three planes only be used at lower resolutions (typically VGA (640×480) or smaller).

The write-back pixel format is always 24-bit RGB 8:8:8. stored in the memory continuously and packed. Figure 10-49 shows the format of the write-back pixels in main memory.

Memory Offset	Local Address [33:35]							
	000	001	010	011	100	101	110	111
0x00	B0	G0	R0	B1	G1	R1	B2	G2
0x08	R2	B3	G3	R3	B4	G4	R4	B5
0x10	G5	R5	B6	G6	R6	B7	G7	R7
0x18	B8	G8	R8	B9	G9	R9	B10	G10
0x20	R10	...	...	...	...	...	...	...

Figure 10-49. Write-Back Pixel Format in Memory

The WB_MEM_ADDR register specifies the address in the memory where the write-back data is stored. Writing to this address triggers a write-back frame refresh. A VSYNC_WB interrupt will occur at the end of a write-back frame, if enabled.

The write-back frame size is specified in the WB_SIZE register (10.3.1.12/10-10).

Note that in mode 2, the write-back frame size must not be bigger than the display size, because the DIU only works when all write-back operations of a frame can complete before the end of the display frame parallel to it. Also note that (DELTA_X_WB × DELTA_Y_WB) must be an integer multiple of 8.

10.4.6 Color Bar Generation

For testing purposes, it is desirable to generate color bars within the DIU itself. This is achieved by setting the DIU_MODE register to 4. The pattern produced is a very simple one with eight vertical color bars. This mode allows the user to verify that the DIU is operational without the need to interact with the system memory. A basic color sequence is preloaded, but the user can overwrite the default values if needed. See Section 10.3.1.20, “COLBAR Registers,” for more information.

Note that programming the color bar registers at the middle of a frame will affect the display immediately, so one should program them after the VSYNC interrupt is detected.

The size of the bars is set by dividing the horizontal resolution by 8 using integer math; if the horizontal resolution is not divisible by 8 exactly, then not all color bars will have the same width.

10.4.7 The Input/Output Pixel Buffer

The DIU possesses an 8-kbyte embedded Dual-Port SRAM used as input/output pixel buffer. It is divided into four 2-kbyte banks. Three banks are used as input buffers for the three input graphic planes; the fourth is used as output buffer for write-back (of output graphics data).

Each 2-kbyte bank is organized as eight 256-byte buffers which are used in a circular fashion.

10.4.8 The Internal DMA

The DIU provides internal direct memory access (DMA), which is designed to transfer data between the main memory and DIU internal memories automatically.

The DMA engine features five physical channels, four to read data from and one to write data back to main memory.

Channels 1~3 are dedicated to transfer input pixel data from main memory into the DIU input pixel buffers, using one channel per plane.When started, they transfer data (for Area Of Interest only, as specified by an AD, see Figure 10-31) from memory to the input buffers line by line, in scanning order. Channel 4 is used in write-back mode only. It transfers blended pixel data back to main memory. Channel 5 is designed to load palette/gamma tables, cursor bitmap and the ADs from main memory into related DIU memories. The DIU internal DMA arbiter determines which channel will be serviced first.

10.4.9 Interrupt Generation

The DIU generates interrupt through a single line that is controlled by the contents of two registers: INT_STATUS (10.3.1.18/10-14) and INT_MASK(10.3.1.19/10-15). When an interrupt occurs, the host needs to read the INT_STATUS register to find the source of the interrupt. The read operation will also clear the register.

There are five interrupt statuses defined, VSYNC -- Vertical Synchronization; VSYNC_WB -- Vertical Synchronization for Write Back; UNDRUN -- Under Run Exception; PARERR -- Display Parameter Error; and LS_BF_VS -- Lines Before VSYNC.

A display parameter error interrupt is generated if the user sets the display parameters wrongly. The following table (Table 10-35) gives a list of parameter error conditions under the different DIU operating modes. When a PARERR interrupt is detected, user can select to turn off the DIU (program the DIU_MODE to 0), correct the wrong parameter(s), and turn it on again. The DIU will be reinitialized internally.

Table 10-35. Parameter Error Conditions

Mode 1 and 2	DELTAXI > DELTAXS
	DELTAYI > DELTAYS
	DELTAXI + OFFSETXD > DELTA_X
	DELTAYI + OFFSETYD > DELTA_Y
	DELTAXI + OFFSETXI > DELTAXS
	DELTAYI + OFFSETYI > DELTAYS

Table 10-35. Parameter Error Conditions (continued)

Mode 2 and 3	DELTAXI > DELTA_X_WB
	DELTAYI > DELTA_Y_WB
	DELTAXI + OFFSETXD > DELTA_X_WB
	DELTAYI + OFFSETYD > DELTA_Y_WB
	DELTAXI + OFFSETXI > DELTA_XS
	DELTAYI + OFFSETYI > DELTA_YS

10.4.10 Dynamic Priority Generation

The multi-port DRAM controller has built-in arbiters that use a 4-bit priority signal (i.e. 16 levels of priority). The DRAM controller attempts to service the request with the highest priority first. The DRAM Controller Priority Manager block, the priority manager, assigns the priority of each task. It dynamically sets the priorities of the DRAM buses in such a way that bandwidth is divided fairly and each master receives its fair share of the bandwidth. Programming the look-up tables in the priority manager allows control of the relative priority to other channels and the average share of bandwidth assigned to the current master.

The DIU outputs a 4-bit priority signal to the priority manager. The priority is a function of the internal input buffer filling level (buffer full means low priority, buffer empty means high priority). The buffer filling level ranges from 0 to 7 and is used to select a LUT component from the PLUT register as the priority. Filling level 0 selects PRIORITY_0 and filling level 7 selects PRIORITY_7.

The priority manager can be programmed to take the DIU priority ouput directly (option 1) or to follow the normal priority schema (option 2). It is recommended to use option 2, the default option.

To use option 1, the user should set PRIOMAN_CONFIG2[DIU-OVERRULE] and program a set of priority values in the PLUT register (PRIORITY_0 to PRIORITY_7 from high to low). The priority value ranges from 0 to 15 and accesses areblocked if the priority value is 0.

To use option 2, it is recommended to program PRIOMAN_CONFIG2[LUT SEL0]=3, in order for the priority manager to select the alternate look-up table for the DIU, if the DIU incoming priority bit 3 is high. Thus, the user can set the DIU priority low in the main look-up table and set it high in the alternate look-up table, and program the PLUT register, so, for example its priority bit 3 is high (0x8) when the buffer is lower than half full. The purpose to do this is to only escalate the DIU priority when necessary, and save the DRAM bandwidth to the other masters like the Power Architecture e300 core.

10.4.11 Display Signal Timing

The first step to generate appropriate timing signals for the selected display is to adjust the frequency of the pixel clock to a frequency that is within the specified parameters of the display (see Section 10.4.11.1, “Refresh Rate). Program the SCFR1 register in the system clock module, SCFR1[DIU_DIV], to set the divide ratio for the DIU pixel clock.

Then the horizontal and vertical synchronize signals are generated based on the pixel clock. The relationship between pixel clock, and HSYNC, VSYNC signals is shown in diagram Figure 10-50 and Figure 10-51.

Refer to Section 10.3.1.11, “DISP_SIZE Register, Section 10.3.1.14, “HSYN_PARA Register, Section 10.3.1.15, “VSYN_PARA Register, Section 10.3.1.16, “SYN_POL Register for related display parameter configuration registers.

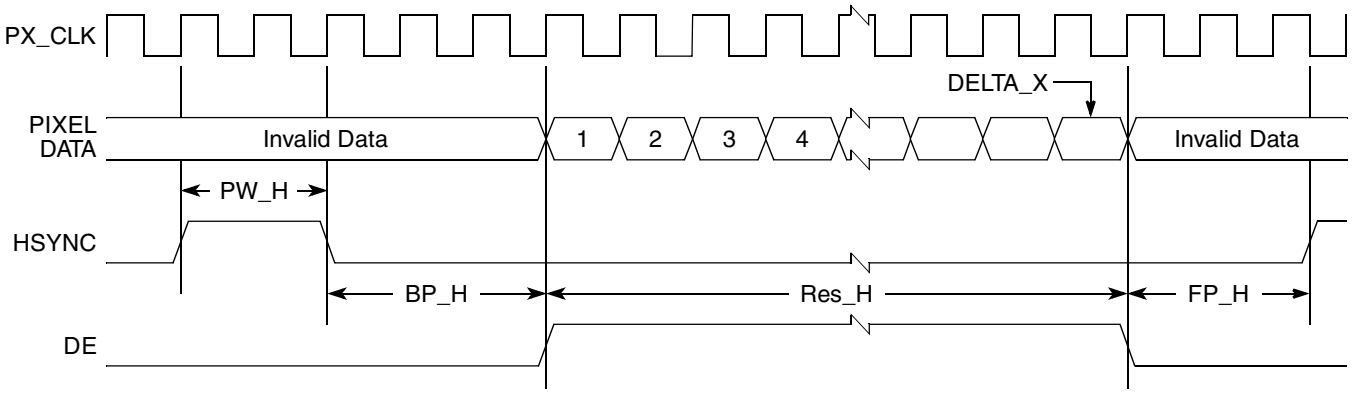


Figure 10-50. Horizontal Sync Signals

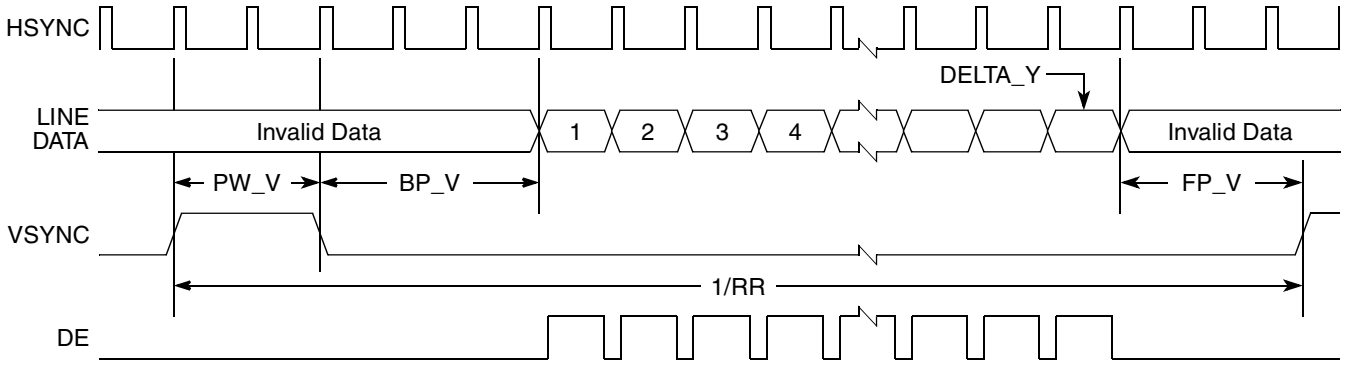


Figure 10-51. Vertical Sync Signals

By default, the DIU outputs (data and sync signals) will switch at same time as the pixel clock rising edge. To provide flexibility in meeting the timing requirements of different LCD display drivers, the user can perform minor tuning of the timing of the pixel clock found on the DIU_CLK_OUT pin relative to all the other DIU signals (DIU_LD[23:0], DIU_VSYNC, DIU_HSYNC, DIU_DE). This phase tuning is done using programmable parameters in the Global Utilities CLKDVDR register, specifically CKDVDR[PXCKINV] and CKDVDR[PXCKDLY]. See Section 23.4.1.18, “Clock Divide Register (CLKDVDR),” for more information. Phase tuning using PXCKINV and PXCKDLY does not change the pixel clocks frequency or duty cycle.

10.4.11.1 Refresh Rate

The refresh rate (or frame rate) is the number of times that the display is updated in a second. It can be calculated from the timing parameters using the following formula:

$$rr = \frac{pix\_clk}{(delta\_x + fp\_h + pw\_h + bp\_h) \times (delta\_y + fp\_v + pw\_v + bp\_v)}$$

Since the user probably has a set target refresh rate (rr), the pix_clk value has been set already and the delta_x and delta_y values are determined exactly by the panel used, the rest of the parameters in this equation must be chosen to approach the desired refresh rate while complying with the requirements established in the panel's data sheet for the front and back porches.

10.5 Initialization/Application Information

10.5.1 DIU Initialization

The following procedure describes how to initialize the DIU. Note that the DIU must not be enabled if the core is in a boot hold off state. Software can determined the boot holdoff state of the core by polling the PORT0_EN bit in the MCM port configuration register (PCR) at CCSR offset 0x0_1010.

1. Hardware reset.
2. Program the display timing signal generation related registers. Failing to set appropriate values for the display timing parameters may result in damage to the display.
3. Prepare the palette/gamma tables, cursor bitmap in the memory and set the pointers to them. Note that these tables should be located in DDR or local bus controller memory; the MPC8610 does not support access to the PCI or PCI Express memory space by the DIU.

   This step is not strictly required because the user might be using it in an application without palette, gamma or cursor. However, it is recommended that the user load zero contents in this case so that the internal SRAM can be initialized.
4. Create the area descriptors (ADs) and set the pointers to them. Note that the ADs should be located in DDR or local bus controller memory; the MPC8610 does not support access to the PCI or PCI Express memory space by the DIU.

   This step is also not strictly required because the default background colors can be displayed or a mode that does not require Area Descriptors may be selected.
5. Program the PLUT register so that the DIU can have the necessary priority (by default, the priority is all zeros, which is the lowest priority level in the system). See Section 10.3.1.22, "PLUT Register," for more information about recommended settings.
6. Program the INT_MASK register and enable the interrupts needed for the application (by default, all interrupts are enabled after hardware reset).
7. Configure the WB_MEM_ADDR register if the operating mode is mode 2 or 3.
8. Set the operating mode (by configuring the DIU_MODE register) to turn on the DIU.

10.5.2 Controlling DIU Planes after the DIU is Enabled

The DIU is initialized by correctly configuring its registers before enabling the DIU by setting DIU_MODE[DIU_MODE] to a legal, non-zero value. The DIU supports up to 3 planes which are:

- activated by setting the corresponding DESC_*n* register to a non-zero value
- deactivated by setting the corresponding DESC_*n* register to all zeros

If the DIU is enabled and a plane is used (the area descriptor pointer DESC_*n* for that plane is non-zero), then that plane must remain in use throughout the time that the DIU is enabled. A value of all zeros must not be written to DESC_*n* as long as the DIU remains enabled (DIU_MODE does not equal 000).

Similarly, if the DIU is enabled and a plane is not used (the area descriptor pointer, DESC_*n*, is all zeros), then that plane must not start to be used after the DIU is enabled. In this scenario, if DESC_*n* is all zeros, then DESC_*n* must not be written as long as the DIU remains enabled (DIU_MODE does not equal 000).

If the capability of adding / removing a plane is desired after the DIU is enabled, this is accomplished using the techniques in the following sections.

10.5.2.1 Activating a Plane after the DIU is Enabled

Once the DIU has been enabled, the user is not permitted to activate a new plane as described above. If there is a possibility of adding an additional plane after the DIU is already enabled, this new plane must be allocated before enabling the DIU by creating this plane with a dummy AOI (see Section 10.5.2.3, "Creating a Dummy Area of Interest," below) which doesn't affect the display and setting its corresponding DESC_*n* register to a non-zero value. The application uses the dummy AOI for this plane until it is ready to activate the plane, which in this case simply means substituting the dummy AOI with a new AOI containing a desired area to display.

10.5.2.2 Deactivating a Plane after the DIU is Enabled

Once the DIU has been enabled, the user is not permitted to deactivate a plane that is currently active. If there is a possibility that the application may need to remove a plane that is currently active after the DIU is already enabled, this is accomplished by creating beforehand a dummy AOI (see Section 10.5.2.3, "Creating a Dummy Area of Interest," below) which doesn't affect the display with its corresponding DESC_*n* register set to a non-zero value. When the application needs to remove a plane, it simply changes the DESC_*n* to point to the dummy AOI, effectively removing the plane because the information in the dummy AOI doesn't affect the display.

10.5.2.3 Creating a Dummy Area of Interest

An example minimum sized dummy AOI can be created using the following settings:

- PIXEL_S=3 (4 bytes per pixel)
- DELTAXS=4, DELTAYS=4
- DELTAXI=2, DELTAYI=4
- OFFSETXI=0, OFFSETYI=0
- OFFSETXD=0, OFFSETYD=0
- Configured for ARGB (8:8:8:8) color format.

When creating a dummy AOI and DIU_MODE = 1 (All three planes output to the display):

- If dummy AOI is in Plane 1:
  Alpha is set to 0x0, RGB is set to the value of the BGND Register (default background color)
- If dummy AOI is in Plane 2 and 3:
  Alpha is set to 0x0, and RGB can be set to any value (plane is transparent)

When creating a dummy AOI and DIU Mode = 2 (Plane 1 to display, Planes 2+3 write back to memory):

- If dummy AOI is in Plane 1:
  Alpha is set to 0x0, RGB is set to the value of the BGND Register (default background color)
- If dummy AOI is in Plane 2:
  Alpha is set to 0x0, RGB is set to the value of the BGND_WB Register (default write back background color)
- If dummy AOI is in Plane 3:
  Alpha is set to 0x0, and RGB can be set to any value (plane is transparent)

When creating a dummy AOI and DIU Mode = 3 (All three planes written back to memory):

- If dummy AOI is in Plane 1:
  Alpha is set to 0x0, and RGB is set to the value of the BGND_WB Register (default write back background color)
- If dummy AOI is in Plane 2 and 3:
  Alpha is set to 0x0, and RGB can be set to any value (plane is transparent)

10.5.3 Synchronizing with the Host

Three interrupt status bits are defined in the DIU for synchronization purposes, they are VSYNC, LS_BF_VS and VSYNC_WB.

If enabled, the VSYNC status bit will always be asserted with the first vsync pulse cycle. With this interrupt, the host can always observe the beginning of a new frame. Beside this, another interrupt, the LS_BF_VS (lines before vsync) can be used to set a deadline for the host to program the pointers for the current next frame. This interrupt is asserted at user-specified lines (set by the LS_BF_VS threshold, Section 10.3.1.17, “THRESHOLDS Register”) before the vertical front porch (FP_V). The host can program the pointer registers after it detects the LS_BF_VS interrupt.

Figure 10-52 shows a diagram on how these two interrupts can be used to synchronize the host and the DIU. In this diagram we have the CPU (e.g. the host), and the DIU. At the end of each frame the DIU begins processing the next frame by first loading the necessary AD, palette, cursor and gamma information from external memory pointed to by its internal register values. All of this takes place in the time frame marked by the dotted blue lines in the diagram below. The host needs to make sure that the proper data is in external memory and that the proper address values are programmed into the DIU’s registers before this time. The time frame marked by the red dotted lines in the diagram is when the host should take care of this. The host should use the LS_BF_VS interrupt to get notified when to begin setting up the DIU for the next display frame. For mode 3, use the VSYNC_WB interrupt. It is clear that the LS_BF_VS threshold should be set to trigger before the FP_V of the current frame (set it greater than 0) or the host will not have enough time to properly program the DIU for the next frame and after the vertical back porch (BP_V) of the previous frame or the host will interfere with the current frame that is to be displayed.

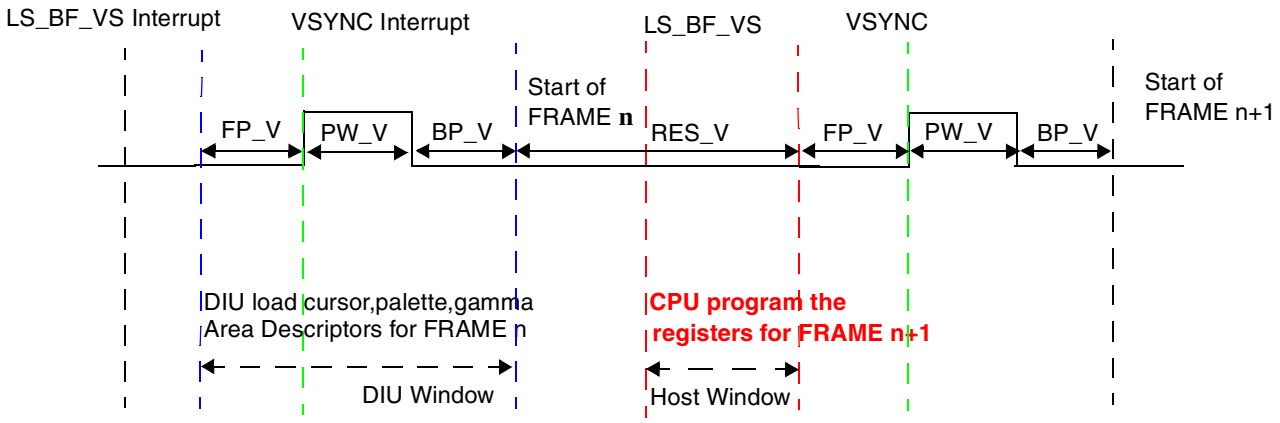


Figure 10-52. Synchronize the Host and the DIU

For the operating modes with write-back operation, a VSYNC_WB interrupt can be used to find the end of a write-back frame. To enable the DIU to work on the next write-back frame, the user must re-program the DESC_1/DESC_2/DESC_3 pointers for the related planes, and the WB_MEM_ADDR register after the VSYNC_WB interrupt is detected.

10.5.4 Recovering from Parameter Errors

A parameter error exception occurs under the conditions detailed in Table 10-35. If enabled, a PARERR interrupt will be issued to the host. On reception of a PARERR interrupt, the user (host) must turn off the DIU, correct the wrong parameters, and then turn it on again (by programming the DIU_MODE register). The DIU will be initialized internally.

Note that when it is turned off, the DIU is initialized immediately regardless of the current bus status, so there could be retractive bus request or pending bus transactions. So if the DIU is going to be turned off separately in an application, the system must support retractive bus requests on the DIU port; it is recommended to wait for some time before turning on the DIU again so that any pending bus transactions can complete on the slave side.

Chapter 11
Programmable Interrupt Controller (PIC)

This chapter describes the programmable interrupt controller (PIC) interrupt protocol, various types of interrupt sources controlled by the PIC, and the PIC registers with some programming guidelines.

11.1 Introduction

The PIC conforms to the OpenPIC architecture. The interrupt controller provides multiprocessor interrupt management, and is responsible for receiving hardware-generated interrupts from different sources (both internal and external), prioritizing them, and delivering them to a CPU for servicing.

11.1.1 Overview

Figure 11-1 is a block diagram showing the relationship of the various functional blocks and how the signals external to the PIC are connected to other blocks on the device, including the cores.

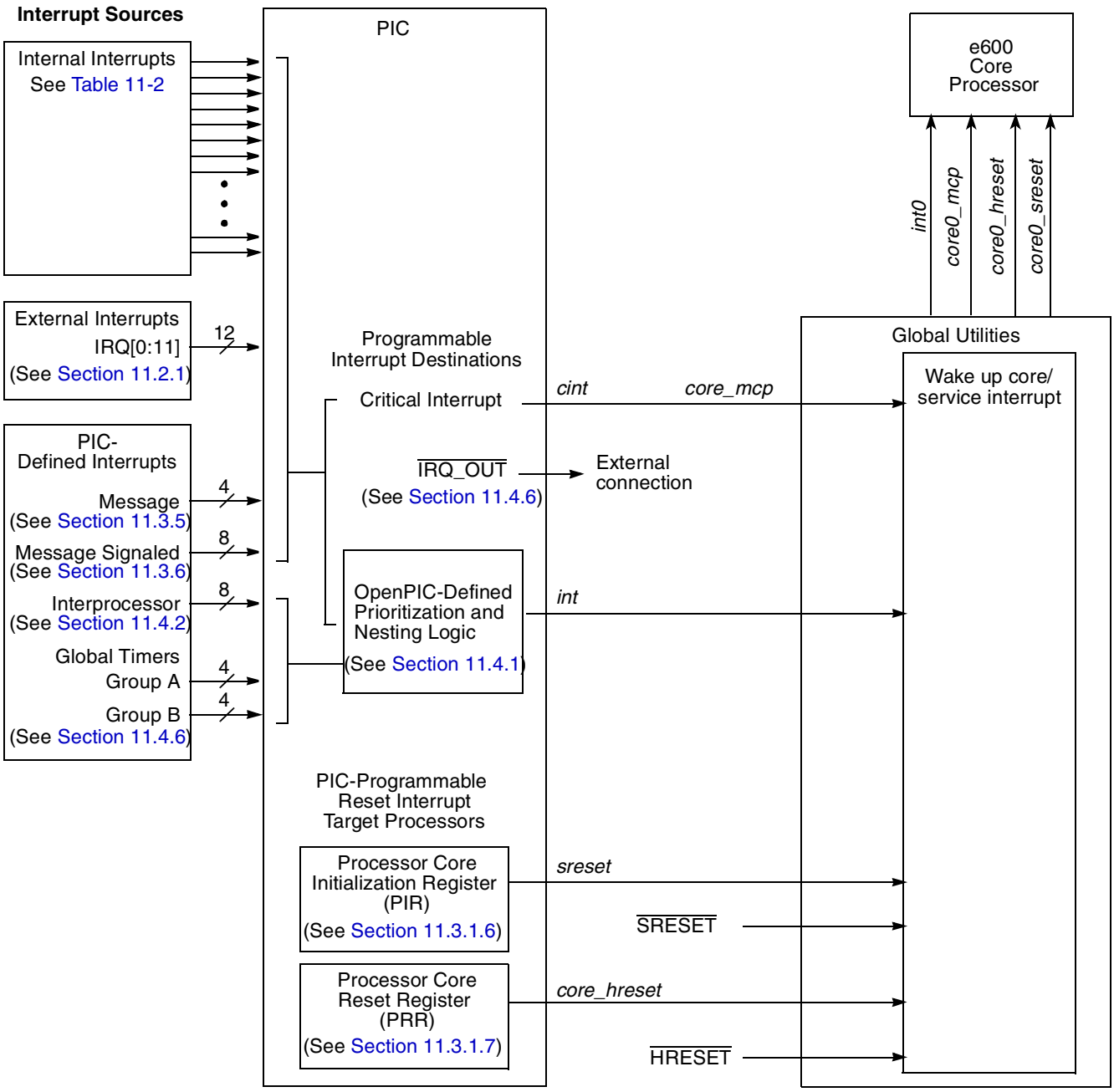


Figure 11-1. Interrupt Sources Block Diagram Features

The PIC has the following features:

- Support for the following interrupt sources:
  - External—Off-chip signals, IRQ[0:11]
  - Internal—These are on-chip sources from peripheral logic within the integrated device signalling error conditions that need to be addressed by software.
- Interrupts generated from within the PIC itself, which are as follows:
  - Global timers A and B internal to the PIC
  - Interprocessor interrupts (IPI)—Intended for communication between different processor cores on the same device. (Can be used for self-interrupt in single-core devices.)
  - Message registers—From within the PIC. Triggered on register write, cleared on read. Used for interprocessor communication.
  - Shared message signaled registers—From within the PIC. Triggered on register write, cleared on read. Used for cross-program communication.
    - Four 32-bit message interrupt channels.
    - Two groups of four global 32-bit timers clocked with the MPX clock or the RTC input. Timers within each group can be concatenated to time longer durations.
- Three types of programmable interrupt outputs:
  - External interrupt (*int*). Any of the PIC interrupt sources can be programmed to direct interrupt requests to *int*. Handling of such interrupt requests follows the OpenPIC specification, which guarantees that the highest priority interrupt supersedes lower priority interrupts. Section 11.4.1.2, “Interrupts Routed to *int*,” describes how the PIC logic handles these interrupts.
  - Critical interrupt (*cint*). Connected to the core’s machine check interrupt input.
  - $\overline{\text{IRQ\_OUT}}$.

    Section 11.4.1.1, “Interrupts Routed to *cint* or IRQ_OUT,” describes how the PIC logic supports this interrupt.
- Programming model compliant with the OpenPIC architecture.
  - Message, interprocessor and global timer interrupts. (Note that the interprocessor and global timer interrupts can only be routed to *int*.)
  - The following OpenPIC-defined features support only interrupts routed to the *int* signal:
    - Fully-nested interrupt delivery, guaranteeing that the interrupt source with the highest priority is given precedence over lower priority interrupts, including any that are in service.
    - 16 programmable interrupt priority levels
    - Support for identifying and handling spurious interrupts
- Support for two processors.
  - Interrupts can be routed to processor core 0 or 1
- Processor core initialization control
- Processor core hardware reset control
- Programmable resetting of the PIC through the global configuration register

- Support for connection of external interrupt controller device such as an 8259 programmable interrupt controller. In 8259 mode, an interrupt causes assertion of a local (that is, internal to the integrated device) interrupt output signal, $\overline{\text{IRQ\_OUT}}$.
- Pass-through mode (PIC disabled) in which the PIC directs interrupts off-chip for external servicing. See Section 11.1.3.2, "Pass-Through Mode (GCR[M] = 0)."

11.1.2 Interrupts to the Processor Core

The external interrupt signal, *int*, is the main interrupt output from the PIC to the processor core.

The interrupt sources can also specify the critical interrupt output, c*int*, if the corresponding *x*IDR*n*[CI0] or *x*IDR*n*[CI1] is set.

The PIC also defines the PIR, described in Section 11.3.1.6, "Processor Core Initialization Register (PIR)," which can be used to reset the core. The PRR, described in Section 11.3.1.7, "Processor Reset Register (PRR)," which can be used to trigger a hard reset. Processor core interrupts generated by the PIC are described in Table 11-1.

Table 11-1. Processor Core Interrupts Generated by the PIC—Types and Sources

Source	Internal External Message Message Signaled Global Timer Interprocessor	Internal External Message Message Signaled		PIR	PRR
PIC output	*int0* or *int1*	*cint0* or *cint1*	$\overline{\text{IRQ\_OUT}}$	*sreset0* or *sreset1*	*core0_hreset* or *core1_hreset*
Description	PIC signals an external interrupt in corresponding core.	Causes machine check condition in corresponding core.	$\overline{\text{IRQ\_OUT}}$ assertion	One of the following: • External $\overline{\text{SRESET0}}$ or $\overline{\text{SRESET1}}$ assertion (and negation) • *sreset0* or *sreset1* output from PIC due to PIR update.	One of the following: • $\overline{\text{HRESET}}$ assertion (and negation) • *core0_hreset* or *core1_hreset* outputs due to PRR update.

11.1.3 Modes of Operation

Mixed or pass-through mode of operation is chosen by setting or clearing GCR[M], as described in Section 11.3.1.4, "Global Configuration Register (GCR)."

11.1.3.1 Mixed Mode (GCR[M] = 1)

In mixed mode, external and internal interrupts are delivered using the normal priority and delivery mechanisms detailed in Section 11.4.1, "Flow of Interrupt Control."

11.1.3.2 Pass-Through Mode (GCR[M] = 0)

The PIC provides a mechanism to support alternate external interrupt controllers such as the PC/AT-compatible 8259 interrupt controller architecture. After a hard reset, the PIC defaults to pass-through mode, in which active-high interrupts from external source IRQ0 are passed directly to core 0 as shown in Figure 11-2; all other external interrupt signals are ignored. Thus, the interrupt signal from an external interrupt controller can be connected to IRQ0 and cause direct interrupts to the processor core 0. The PIC does not perform a vector fetch from an 8259 interrupt controller.

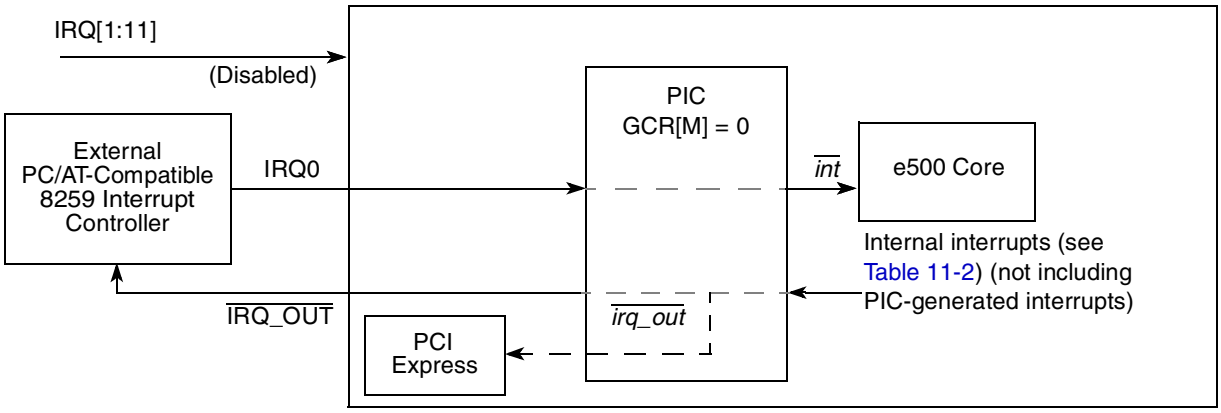


Figure 11-2. Pass-Through Mode Example

When pass-through mode is enabled, the internally-generated interrupts shown in Table 11-3 are not forwarded to core 0. Instead, the PIC passes the raw interrupts from the internal sources to $\overline{\text{IRQ\_OUT}}$. Note that when the PCI Express controller is configured as an endpoint (EP) device, the *irq_out* signal from the PIC may used to automatically generate an outbound PCI Express MSI transaction toward the remote interrupt controller resource on the root complex (RC). See Section 21.4.2.1.2, “Hardware MSI Generation.”

Note that in pass-through mode, interrupts generated within the PIC (global timers, interprocessor, and message register interrupts) are disabled. If internal or PIC-generated interrupts must be reported internally to the processor, mixed mode must be used.

It is required that in pass-through mode the internal and external interrupt targets should be *int*, which is by default.

11.1.4 Interrupt Sources

The PIC can receive separate interrupts from the following sources:

- External—Off-chip signals, IRQ[0:11]
- Internal—On-chip sources from peripheral logic within the integrated device. See Table 11-2.
- Global timers A and B internal to the PIC
- Interprocessor interrupts (IPI)—Intended for communication between different processor cores on the same device. (Can be used for self-interrupt in single-core implementations.)

- Message registers—From within the PIC. Triggered on register write, cleared on read. Used for interprocessor communication.
- Shared message signaled registers—From within the PIC. Triggered on register write, cleared on read. Used for cross-program communication.

11.1.4.1 Interrupt Routing—Mixed Mode

When an interrupt request is delivered to the PIC, the corresponding interrupt destination register is checked to determine where the request should be routed, as follows:

- If *x*IDR*n*[EP] = 1 (and all other destination bits are zero), the interrupt is routed off-chip to the external $\overline{\text{IRQ\_OUT}}$ signal. Or if the PCI Express controller is in EP mode and automatically generates a PCI Express MSI transaction. See Section 21.4.2.1.2, “Hardware MSI Generation.”
- If *x*IDR[CI] is set (and all other destination bits are zero), the interrupt is routed to *cint*. .
- If *x*IDR*n*[P0] is set (and all other destination bits are zero) the interrupt is routed to *int0*. Setting *x*IDR*n*[P1] likewise routs the interrupt to *int1* In this case, the interrupt is latched by the interrupt pending register (IPR) and the interrupt flow is as described in Section 11.4.1, “Flow of Interrupt Control.”

11.1.4.2 Interrupt Destinations

Following its reset (by default), the PIC directs all timer, shared message signaled, and interrupts from external and internal sources to *int* output (connected to the *int* signal of the processor core).

All other interrupts have more destination options, but only one destination can be chosen for a single interrupt. Instead of being routed to *int*, these interrupts can be routed to the core through $\overline{\text{IRQ\_OUT}}$ or *cint*. These options are selected by writing to the EP or CI fields in the appropriate destination register. The global utilities block routes the critical interrupt to *core_mcp*.

11.1.4.3 Internal Interrupt Sources

Table 11-2 shows the assignments of the internal interrupt sources and how they are mapped to the registers that control them. Only the internal interrupts used are listed; that is, the numbers are not consecutive.

Table 11-2. Internal Interrupt Assignments

Internal Interrupt Number	Interrupt Source
0	Reserved
1	MCM
2	DDR Memory Controller
3	eLBC
4	DMA1 Channel 0
5	DMA1 Channel 1
6	DMA1 Channel 2

Table 11-2. Internal Interrupt Assignments (continued)

Internal Interrupt Number	Interrupt Source
7	DMA1 Channel 3
8	PCI
9	PCI Express 8 Lane
10	PCI Express 4 Lane
11	Reserved
12	DUART2
13–25	Reserved
26	DUART1
27	I2C
28	Performance Monitor
29–30	Reserved
31	GPIO Ports
32–42	Reserved
43	SPI
44–45	Reserved
46	SSI1
47	SSI2
48–55	Reserved
56	LCD Controller
57	Infrared Modules
58	GTMs
59	Reserved
60	DMA2 Channel 0
61	DMA2 Channel 1
62	DMA2 Channel 2
63	DMA2 Channel 3

11.2 External Signal Descriptions

The following sections describe the PIC signals.

11.2.1 Signal Overview

The PIC external interface signals are described in Table 11-3. There are 12 distinct external interrupt request input signals and 1 interrupt request output signal IRQ_OUT. As Table 11-3 shows, 8 of the IRQ inputs are also used for delivering INTx signals for the PCI Express root complexes.

11.2.2 Detailed Signal Descriptions

Table 11-3 provides detailed descriptions of the external PIC signals.

Table 11-3. Interrupt Signals—Detailed Signal Descriptions

Signal	I/O	Description	
IRQ[0:11]	I	Interrupt request 0–11. The polarity and sense of each of these signals is programmable. All of these inputs can be driven asynchronously. **Note:** Some interrupt request signals IRQ*n* may share PIC external interrupt registers with PCI Express INTx signaling. See Section 11.4.5, "PCI Express INTx/IRQn Sharing."	
		State Meaning	Asserted—When an external interrupt signal is asserted (according to the programmed polarity), the PIC checks its priority and the interrupt is conditionally passed to the processor designated in the corresponding destination register. In pass-through mode, only interrupts detected on IRQ0 are passed directly to core 0. Negated—There is no incoming interrupt from that source.
		Timing	Assertion—All of these inputs can be asserted asynchronously. Negation—Interrupts programmed as level-sensitive must remain asserted until serviced. Timing requirements for edge-sensitive interrupts can be found in the *Hardware Specifications*.
IRQ_OUT	O	Interrupt request out. When the PIC is programmed in pass-through mode, this output reflects the raw interrupts generated by on-chip sources. See Section 11.1.3, "Modes of Operation."	
		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 occurs asynchronously with respect to the interrupt source. All timing given here is approximate. Assertion—Internal interrupt source: 2 MPX clock cycles after interrupt occurs. External interrupt source: 4 cycles after interrupt occurs. Message interrupts: 2 cycles after write to message register. Negation—Follows interrupt source negation with the following delay: Internal interrupt: 2 MPX clock cycles External interrupt: 4 cycles. Message interrupts: 2 cycles after message register cleared.
MCP	I	Machine check processor. Assertion causes a machine check interrupt to the core. Note that if the core is not configured to process machine check interrupts (MSR[ME] = 0), assertion of MCP causes a core checkstop condition. Note that internal sources for the internal *core_mcp* can also cause a machine check interrupt to the processor core as described in Section 23.4.1.13, "Machine Check Summary Register (MCPSUMR)," and Table 5-2.	
		State Meaning	Asserted—Integrated logic should direct the core to take a machine check interrupt or enter the checkstop state as directed by the MSR. Negated—Machine check handling is not being requested by the external system.
		Timing	Assertion—May occur at any time, asynchronous to any clock. Negation—Because MCP*n* is edge-triggered, it can be negated one clock after its assertion.

11.3 Memory Map/Register Definition

The PIC programmable register map occupies 256 Kbytes of memory-mapped space. Reading undefined portions of the memory map returns all zeros; writing has no effect. All PIC registers are 32 bits wide and, although located on 128-bit address boundaries, should be accessed only as 32-bit quantities.

The PIC address offset map, shown in Table 11-4, is divided into three areas:

- 0x*nn*4_0000–0x*nn*4_FFF0—Global registers
- 0x*nn*5_0000–0x*nn*5_FFF0—Interrupt source configuration registers
- 0x*nn*6_0000–0x*nn*6_FFF0—Per-CPU registers

Table 11-4. PIC Register Address Map

Offset	Register	Access	Reset	Section/Page
Global Registers—Block Base Address: 0x4_0000				
0x0000	BRR1—Block revision register 1	R	0x0040_0300	11.3.1.1/11-18
0x0010	BRR2—Block revision register 2	R	0x0000_0001	11.3.1.2/11-19
0x0014–0x003F	Reserved	—	—	—
0x0040	IPIDR0—Interprocessor interrupt 0 (IPI 0) dispatch register	W	All zeros	11.3.8.1/11-49
0x0050	IPIDR1—IPI 1 dispatch register			
0x0060	IPIDR2—IPI 2 dispatch register			
0x0070	IPIDR3—IPI 3 dispatch register			
0x0080	CTPR—Current task priority register	R/W	0x0000_000F	11.3.8.2/11-49
0x0090	WHOAMI—Who am I register	R	0x0000_00*nn*	11.3.8.3/11-50
0x00A0	IACK—Interrupt acknowledge register	R	All zeros	11.3.8.4/11-51
0x00B0	EOI—End of interrupt register	W	All zeros	11.3.8.5/11-51
0x00B4–0x0FFF	Reserved	—	—	—
0x1000	FRR—Feature reporting register	R	0x0067_0002	11.3.1.3/11-19
0x1004–0x101F	Reserved	—	—	—
0x1020	GCR—Global configuration register	R/W	All zeros	11.3.1.4/11-20
0x1024–0x1003F	Reserved	—	—	—
0x1040–0x107F	Vendor reserved	—	—	—
0x1080	VIR—Vendor identification register	R	All zeros	11.3.1.5/11-21
0x1090	PIR—Processor core initialization register	R/W	All zeros	11.3.1.6/11-21
0x1098	PRR—Processor reset register	R/W	All zeros	11.3.1.7/11-22

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x10A0	IPIVPR0—IPI 0 vector/priority register	R/W	0x8000_0000	11.3.1.8/11-22
0x10B0	IPIVPR1—IPI 1 vector/priority register			
0x10C0	IPIVPR2—IPI 2 vector/priority register			
0x10D0	IPIVPR3—IPI 3 vector/priority register			
0x10E0	SVR—Spurious vector register	R/W	0x0000_FFFF	11.3.1.9/11-23
Global Timer Group A Registers				
0x10F0	TFRRA—Timer frequency reporting register (Group A)	R/W	All zeros	11.3.2.1/11-24
0x1100	GTCCRA0—Global timer 0 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x1110	GTBCRA0—Global timer 0 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25
0x1120	GTVPRA0—Global timer 0 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x1130	GTDRA0—Global timer 0 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x1140	GTCCRA1—Global timer 1 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x1150	GTBCRA1—Global timer 1 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25
0x1160	GTVPRA1—Global timer 1 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x1170	GTDRA1—Global timer 1 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x1180	GTCCRA2—Global timer 2 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x1190	GTBCRA2—Global timer 2 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25
0x11A0	GTVPRA2—Global timer 2 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x11B0	GTDRA2—Global timer 2 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x11C0	GTCCRA3—Global timer 3 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x11D0	GTBCRA3—Global timer 3 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25
0x11E0	GTVPRA3—Global timer 3 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x11F0	GTDRA3—Global timer 3 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x11F4–0x12FF	Reserved	—	—	—
0x1300	TCRA—Timer control register (Group A)	R/W	All zeros	11.3.2.6/11-27
0x1308	ERQSR—External interrupt summary register	R	All zeros	11.3.3.1/11-29
0x1310	IRQSR0—IRQ_OUT summary register 0	R	All zeros	11.3.3.2/11-30
0x1320	IRQSR1—IRQ_OUT summary register 1	R	All zeros	11.3.3.3/11-31
0x1324	IRQSR2—IRQ_OUT summary register 2	R	All zeros	11.3.3.4/11-31
0x1330	CISR0—Critical interrupt summary register 0	R	All zeros	11.3.3.5/11-32
0x1340	CISR1—Critical interrupt summary register 1	R	All zeros	11.3.3.6/11-33
0x1344	CISR2—Critical interrupt summary register 2	R	All zeros	11.3.3.7/11-33
0x1350	PM0MR0—Performance monitor 0 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34
0x1360	PM0MR1—Performance monitor 0 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1364	PM0MR2—Performance monitor 0 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1370	PM1MR0—Performance monitor 1 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x1380	PM1MR1—Performance monitor 1 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1384	PM1MR2—Performance monitor 1 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1390	PM2MR0—Performance monitor 2 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34
0x13A0	PM2MR1—Performance monitor 2 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13A4	PM2MR2—Performance monitor 2 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13B0	PM3MR0—Performance monitor 3 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34
0x13C0	PM3MR1—Performance monitor 3 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13C4	PM3MR2—Performance monitor 3 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13C8–0x13FF	Reserved	—	—	—
0x1400	MSGR0—Message register 0	R/W	All zeros	11.3.5.1/11-36
0x1410	MSGR1—Message register 1	R/W	All zeros	11.3.5.1/11-36
0x1420	MSGR2—Message register 2	R/W	All zeros	11.3.5.1/11-36
0x1430	MSGR3—Message register 3	R/W	All zeros	11.3.5.1/11-36
0x1434–0x14FF	Reserved	—	—	—
0x1500	MER—Message enable register	R/W	All zeros	11.3.5.2/11-36
0x1510	MSR—Message status register	R/W	All zeros	11.3.5.3/11-37
0x1514–0x15FF	Reserved	—	—	—
0x1600	MSIR0—Shared message signaled interrupt register 0	RC	All zeros	11.3.6.1/11-37
0x1610	MSIR1—Shared message signaled interrupt register 1	RC	All zeros	11.3.6.1/11-37
0x1620	MSIR2—Shared message signaled interrupt register 2	RC	All zeros	11.3.6.1/11-37
0x1630	MSIR3—Shared message signaled interrupt register 3	RC	All zeros	11.3.6.1/11-37
0x1640	MSIR4—Shared message signaled interrupt register 4	RC	All zeros	11.3.6.1/11-37
0x1650	MSIR5—Shared message signaled interrupt register 5	RC	All zeros	11.3.6.1/11-37
0x1660	MSIR6—Shared message signaled interrupt register 6	RC	All zeros	11.3.6.1/11-37
0x1670	MSIR7—Shared message signaled interrupt register 7	RC	All zeros	11.3.6.1/11-37
0x1674–0x171F	Reserved	—	—	—
0x1720	MSISR—Shared message signaled interrupt status register	R	All zeros	11.3.6.2/11-38
0x1740	MSIIR—Shared message signaled interrupt index register	W	All zeros	11.3.6.3/11-38
0x1744–0x20EF	Reserved	—	—	—
Global Timer Group B Registers				
0x20F0	TFRRB—Timer frequency reporting register group B	R/W	All zeros	11.3.2.1/11-24
0x2100	GTCCRB0—Global timer current count register group B 0	R	All zeros	11.3.2.2/11-25
0x2110	GTBCRB0—Global timer base count register group B 0	R/W	0x8000_0000	11.3.2.3/11-25

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x2120	GTVPRB0—Global timer vector/priority register group B 0	R/W	0x8000_0000	11.3.2.4/11-26
0x2130	GTDRB0—Global timer destination register group B 0	R/W	0x0000_0001	11.3.2.5/11-27
0x2140	GTCCRB1—Global timer current count register group B 1	R	All zeros	11.3.2.2/11-25
0x2150	GTBCRB1—Global timer base count register group B 1	R/W	0x8000_0000	11.3.2.3/11-25
0x2160	GTVPRB1—Global timer vector/priority register group B 1	R/W	0x8000_0000	11.3.2.4/11-26
0x2170	GTDRB1—Global timer destination register group B 1	R/W	0x0000_0001	11.3.2.5/11-27
0x2180	GTCCRB2—Global timer current count register group B 2	R	All zeros	11.3.2.2/11-25
0x2190	GTBCRB2—Global timer base count register group B 2	R/W	0x8000_0000	11.3.2.3/11-25
0x21A0	GTVPRB2—Global timer vector/priority register group B 2	R/W	0x8000_0000	11.3.2.4/11-26
0x21B0	GTDRB2—Global timer destination register group B 2	R/W	0x0000_0001	11.3.2.5/11-27
0x21C0	GTCCRB3—Global timer current count register group B 3	R	All zeros	11.3.2.2/11-25
0x21D0	GTBCRB3—Global timer base count register group B 3	R/W	0x8000_0000	11.3.2.3/11-25
0x21E0	GTVPRB3—Global timer vector/priority register group B 3	R/W	0x8000_0000	11.3.2.4/11-26
0x21F0	GTDRB3—Global timer destination register group B 3	R/W	0x0000_0001	11.3.2.5/11-27
0x21F4–0x22FF	Reserved	—	—	—
0x2300	TCRB—Timer control register (Group B)	R/W	All zeros	11.3.2.6/11-27
0x2304–0xFFF0	Reserved	—	—	—
Interrupt Source Configuration Registers—Block Base Address: 0x5_0000				
0x0000	EIVPR0—External interrupt 0 (IRQ0) vector/priority register or PEX1-INTA vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0010	EIDR0—External interrupt 0 (IRQ0) destination register or PEX1-INTA destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0020	EIVPR1—External interrupt 1 (IRQ1) vector/priority register or PEX1-INTB vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0030	EIDR1—External interrupt 1 (IRQ1) destination register or PEX1-INTB destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0040	EIVPR2—External interrupt 2 (IRQ2) vector/priority register or PEX1-INTC vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0050	EIDR2—External interrupt 2 (IRQ2) destination register or PEX1-INTC destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0060	EIVPR3—External interrupt 3 (IRQ3) vector/priority register or PEX1-INTD vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0070	EIDR3—External interrupt 3 (IRQ3) destination register or PEX1-INTD destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0080	EIVPR4—External interrupt 4 (IRQ4) vector/priority register or PEX2-INTA vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0090	EIDR4—External interrupt 4 (IRQ4) destination register or PEX2-INTA destination register	R/W	0x0000_0001	11.3.7.2/11-43

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x00A0	EIVPR5—External interrupt 5 (IRQ5) vector/priority register or PEX2-INTB vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x00B0	EIDR5—External interrupt 5 (IRQ5) destination register or PEX2-INTB destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x00C0	EIVPR6—External interrupt 6 (IRQ6) vector/priority register or PEX2-INTC vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x00D0	EIDR6—External interrupt 6 (IRQ6) destination register or PEX2-INTC destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x00E0	EIVPR7—External interrupt 7 (IRQ7) vector/priority register or PEX2-INTD vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x00F0	EIDR7—External interrupt 7 (IRQ7) destination register or PEX2-INTD destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0100	EIVPR8—External interrupt 8 (IRQ8) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0110	EIDR8—External interrupt 8 (IRQ8) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0120	EIVPR9—External interrupt 9 (IRQ9) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0130	EIDR9—External interrupt 9 (IRQ9) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0140	EIVPR10—External interrupt 10 (IRQ10) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0150	EIDR10—External interrupt 10 (IRQ10) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0160	EIVPR11—External interrupt 11 (IRQ11) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0170	EIDR11—External interrupt 11 (IRQ11) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0174– 0x01FF	Reserved	—	—	—
0x0200	IIVPR0—Internal interrupt 0 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0210	IIDR0—Internal interrupt 0 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0220	IIVPR1—Internal interrupt 1 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0230	IIDR1—Internal interrupt 1 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0240	IIVPR2—Internal interrupt 2 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0250	IIDR2—Internal interrupt 2 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0260	IIVPR3—Internal interrupt 3 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0270	IIDR3—Internal interrupt 3 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0280	IIVPR4—Internal interrupt 4 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0290	IIDR4—Internal interrupt 4 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x02A0	IIVPR5—Internal interrupt 5 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x02B0	IIDR5—Internal interrupt 5 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x02C0	IIVPR6—Internal interrupt 6 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x02D0	IIDR6—Internal interrupt 6 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x02E0	IIVPR7—Internal interrupt 7 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x02F0	IIDR7—Internal interrupt 7 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0300	IIVPR8—Internal interrupt 8 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0310	IIDR8—Internal interrupt 8 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0320	IIVPR9—Internal interrupt 9 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0330	IIDR9—Internal interrupt 9 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0340	IIVPR10—Internal interrupt 10 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0350	IIDR10—Internal interrupt 10 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0360	IIVPR11—Internal interrupt 11 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0370	IIDR11—Internal interrupt 11 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0380	IIVPR12—Internal interrupt 12 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0390	IIDR12—Internal interrupt 12 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x03A0	IIVPR13—Internal interrupt 13 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x03B0	IIDR13—Internal interrupt 13 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x03C0	IIVPR14—Internal interrupt 14 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x03D0	IIDR14—Internal interrupt 14 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x03E0	IIVPR15—Internal interrupt 15 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x03F0	IIDR15—Internal interrupt 15 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0400	IIVPR16—Internal interrupt 16 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0410	IIDR16—Internal interrupt 16 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0420	IIVPR17—Internal interrupt 17 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0430	IIDR17—Internal interrupt 17 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0440	IIVPR18—Internal interrupt 18 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0450	IIDR18—Internal interrupt 18 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0460	IIVPR19—Internal interrupt 19 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0470	IIDR19—Internal interrupt 19 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0480	IIVPR20—Internal interrupt 20 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0490	IIDR20—Internal interrupt 20 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x04A0	IIVPR21—Internal interrupt 21 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x04B0	IIDR21—Internal interrupt 21 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x04C0	IIVPR22—Internal interrupt 22 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x04D0	IIDR22—Internal interrupt 22 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x04E0	IIVPR23—Internal interrupt 23 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x04F0	IIDR23—Internal interrupt 23 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0500	IIVPR24—Internal interrupt 24 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0510	IIDR24—Internal interrupt 24 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0520	IIVPR25—Internal interrupt 25 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0530	IIDR25—Internal interrupt 25 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0540	IIVPR26—Internal interrupt 26 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0550	IIDR26—Internal interrupt 26 destination register	R/W	0x0000_0001	11.3.7.4/11-45

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0560	IIVPR27—Internal interrupt 27 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0570	IIDR27—Internal interrupt 27 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0580	IIVPR28—Internal interrupt 28 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0590	IIDR28—Internal interrupt 28 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x05A0	IIVPR29—Internal interrupt 29 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x05B0	IIDR29—Internal interrupt 29 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x05C0	IIVPR30—Internal interrupt 30 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x05D0	IIDR30—Internal interrupt 30 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x05E0	IIVPR31—Internal interrupt 31 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x05F0	IIDR31—Internal interrupt 31 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0600	IIVPR32—Internal interrupt 32 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0610	IIDR32—Internal interrupt 32 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0620	IIVPR33—Internal interrupt 33 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0630	IIDR33—Internal interrupt 33 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0640	IIVPR34—Internal interrupt 34 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0650	IIDR34—Internal interrupt 34 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0660	IIVPR35—Internal interrupt 35 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0670	IIDR35—Internal interrupt 35 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0680	IIVPR36—Internal interrupt 36 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0690	IIDR36—Internal interrupt 36 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x06A0	IIVPR37—Internal interrupt 37 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x06B0	IIDR37—Internal interrupt 37 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x06C0	IIVPR38—Internal interrupt 38 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x06D0	IIDR38—Internal interrupt 38 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x06E0	IIVPR39—Internal interrupt 39 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x06F0	IIDR39—Internal interrupt 39 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0700	IIVPR40—Internal interrupt 40 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0710	IIDR40—Internal interrupt 40 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0720	IIVPR41—Internal interrupt 41 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0730	IIDR41—Internal interrupt 41 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0740	IIVPR42—Internal interrupt 42 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0750	IIDR42—Internal interrupt 42 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0760	IIVPR43—Internal interrupt 43 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0770	IIDR43—Internal interrupt 43 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0780	IIVPR44—Internal interrupt 44 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0790	IIDR44—Internal interrupt 44 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x07A0	IIVPR45—Internal interrupt 45 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x07B0	IIDR45—Internal interrupt 45 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x07C0	IIVPR46—Internal interrupt 46 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x07D0	IIDR46—Internal interrupt 46 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x07E0	IIVPR47—Internal interrupt 47 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x07F0	IIDR47—Internal interrupt 47 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x07F4–0x15F0	Reserved	—	—	—
0x0800	IIVPR48—Internal interrupt 48 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0810	IIDR48—Internal interrupt 48 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0820	IIVPR49—Internal interrupt 49 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0830	IIDR49—Internal interrupt 49 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0840	IIVPR50—Internal interrupt 50 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0850	IIDR50—Internal interrupt 50 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0860	IIVPR51—Internal interrupt 51 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0870	IIDR51—Internal interrupt 51 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0880	IIVPR52—Internal interrupt 52 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0890	IIDR52—Internal interrupt 52 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x08A0	IIVPR53—Internal interrupt 53 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x08B0	IIDR53—Internal interrupt 53 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x08C0	IIVPR54—Internal interrupt 54 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x08D0	IIDR54—Internal interrupt 54 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x08E0	IIVPR55—Internal interrupt 55 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x08F0	IIDR55—Internal interrupt 55 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0900	IIVPR56—Internal interrupt 56 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0910	IIDR56—Internal interrupt 56 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0920	IIVPR57—Internal interrupt 57 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0930	IIDR57—Internal interrupt 57 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0940	IIVPR58—Internal interrupt 58 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0950	IIDR58—Internal interrupt 58 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0960	IIVPR59—Internal interrupt 59 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0970	IIDR59—Internal interrupt 59 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0980	IIVPR60—Internal interrupt 60 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0990	IIDR60—Internal interrupt 60 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x09A0	IIVPR61—Internal interrupt 61 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x09B0	IIDR61—Internal interrupt 61 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x09C0	IIVPR62—Internal interrupt 62 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x09D0	IIDR62—Internal interrupt 62 destination register	R/W	0x0000_0001	11.3.7.4/11-45

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x09E0	IIVPR63—Internal interrupt 63 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x09F0	IIDR63—Internal interrupt 63 destination register	R/W	0x0000_0001	11.3.7.3/11-44
0x1600	MIVPR0—Messaging interrupt 0 (MSG 0) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46
0x1610	MIDR0—Messaging interrupt 0 (MSG 0) destination register	R/W	0x0000_0001	11.3.7.6/11-46
0x1620	MIVPR1—Messaging interrupt 1 (MSG 1) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46
0x1630	MIDR1—Messaging interrupt 1 (MSG 1) destination register	R/W	0x0000_0001	11.3.7.6/11-46
0x1640	MIVPR2—Messaging interrupt 2 (MSG 2) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46
0x1650	MIDR2—Messaging interrupt 2 (MSG 2) destination register	R/W	0x0000_0001	11.3.7.6/11-46
0x1660	MIVPR3—Messaging interrupt 3 (MSG 3) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46
0x1670	MIDR3—Messaging interrupt 3 (MSG 3) destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x1674–0x1BF0	Reserved	—	—	—
0x1C00	MSIVPR0—Shared message signaled interrupt vector/priority register 0	R/W	0x8000_0000	11.3.6.4/11-39
0x1C10	MSIDR0—Shared message signaled interrupt destination register 0	R/W	0x0000_0001	11.3.6.5/11-40
0x1C20	MSIVPR1—Shared message signaled interrupt vector/priority register 1	R/W	0x8000_0000	11.3.6.4/11-39
0x1C30	MSIDR1—Shared message signaled interrupt destination register 1	R/W	0x0000_0001	11.3.6.5/11-40
0x1C40	MSIVPR2—Shared message signaled interrupt vector/priority register 2	R/W	0x8000_0000	11.3.6.4/11-39
0x1C50	MSIDR2—Shared message signaled interrupt destination register 2	R/W	0x0000_0001	11.3.6.5/11-40
0x1C60	MSIVPR3—Shared message signaled interrupt vector/priority register 3	R/W	0x8000_0000	11.3.6.4/11-39
0x1C70	MSDIR3—Shared message signaled interrupt destination register 3	R/W	0x0000_0001	11.3.6.5/11-40
0x1C80	MSIVPR4—Shared message signaled interrupt vector/priority register 4	R/W	0x8000_0000	11.3.6.4/11-39
0x1C90	MSIDR4—Shared message signaled interrupt destination register 4	R/W	0x0000_0001	11.3.6.5/11-40
0x1CA0	MSIVPR5—Shared message signaled interrupt vector/priority register 5	R/W	0x8000_0000	11.3.6.4/11-39
0x1CB0	MSIDR5—Shared message signaled interrupt destination register 5	R/W	0x0000_0001	11.3.6.5/11-40
0x1CC0	MSIVPR6—Shared message signaled interrupt vector/priority register 6	R/W	0x8000_0000	11.3.6.4/11-39
0x1CD0	MSIDR6—Shared message signaled interrupt destination register 6	R/W	0x0000_0001	11.3.6.5/11-40
0x1CE0	MSIVPR7—Shared message signaled interrupt vector/priority register 7	R/W	0x8000_0000	11.3.6.4/11-39
0x1CF0	MSDIR7—Shared message signaled interrupt destination register 7	R/W	0x0000_0001	11.3.6.5/11-40
0x1CFF–0xFFFF	Reserved	—	—	—
Per-CPU Registers Block Base Address: 0x6_0000				
0x0000–0x003F	Reserved	—	—	—
0x0040	IPIDR0—Processor core 0 interprocessor 0 dispatch register	W	All zeros	11.3.8.1/11-49
0x0050	IPIDR1—Processor core 0 interprocessor 1 dispatch register			
0x0060	IPIDR2—Processor core 0 interprocessor 2 dispatch register			
0x0070	IPIDR3—Processor core 0 interprocessor 3 dispatch register			
0x0080	CTPR0—Processor core 0 current task priority register	R/W	0x0000_000F	11.3.8.2/11-49

Table 11-4. PIC Register Address Map (continued)

Offset	Register	Access	Reset	Section/Page
0x0090	WHOAMI0—Processor core 0 who am I register	R	0x0000_00*nn*	11.3.8.3/11-50
0x00A0	IACK0—Processor core 0 interrupt acknowledge register	R	All zeros	11.3.8.4/11-51
0x00B0	EOI0—Processor core 0 end of interrupt register	W	All zeros	11.3.8.5/11-51
0x00BF–0x0FFF	Reserved	—	—	—
0x1000–0x103F	Reserved	—	—	—
0x1040	IPIDR0—Processor core 1 interprocessor 0 dispatch register	W	All zeros	11.3.8.1/11-49
0x1050	IPIDR1—Processor core 1 interprocessor 1 dispatch register			
0x1060	IPIDR2—Processor core 1 interprocessor 2 dispatch register			
0x1070	IPIDR3—Processor core 1 interprocessor 3 dispatch register			
0x1080	CTPR1—Processor core 1 current task priority register	R/W	0x0000_000F	11.3.8.2/11-49
0x1090	WHOAMI1—Processor core 1 who am I register	R	n/a	11.3.8.3/11-50
0x10A0	IACK1—Processor core 1 interrupt acknowledge register	R	All zeros	11.3.8.4/11-51
0x10B0	EOI1—Processor core 1 end of interrupt register	W	All zeros	11.3.8.5/11-51
0x6_10BF–0x6_FFFF	Reserved	—	—	—

11.3.1 Global Registers

Although most PIC registers have one address, some are replicated for each processor core in a multiprocessor device. For such registers, each core accesses its separate registers using the same address, the address decoding being sensitive to the processor core ID. A copy of the per-CPU registers is available to each processor core at the same physical address, that is, in a private access address space that acts like an alias to a processor's own copy of the per-CPU registers. As shown in Figure 11-45, the ID of the core initiating the read/write transaction determines which processor's per-CPU registers to access. For more information, see Section 11.3.8, "Per-CPU (Private Access) Registers."

NOTE

Register fields designated as write-1-to-clear are cleared only by writing ones to them. Writing zeros to them has no effect.

11.3.1.1 Block Revision Register 1 (BRR1)

BRR1, shown in Figure 11-3, provides information about the PIC IP block.

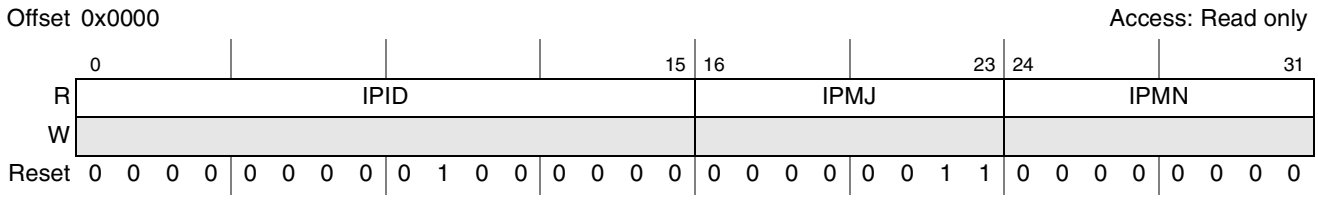


Figure 11-3. Block Revision Register 1 (BRR1)

Table 11-7 describes the BRR1 fields.

Table 11-5. BRR1 Field Descriptions

Bits	Name	Description
0–15	IPID	IP block ID.
16–23	IPMJ	The major revision of the IP block.
24–31	IPMN	The minor revision of the IP block.

11.3.1.2 Block Revision Register 2 (BRR2)

BRR2, shown in Figure 11-4, provides information about the IP block integration option and IP block configuration options.

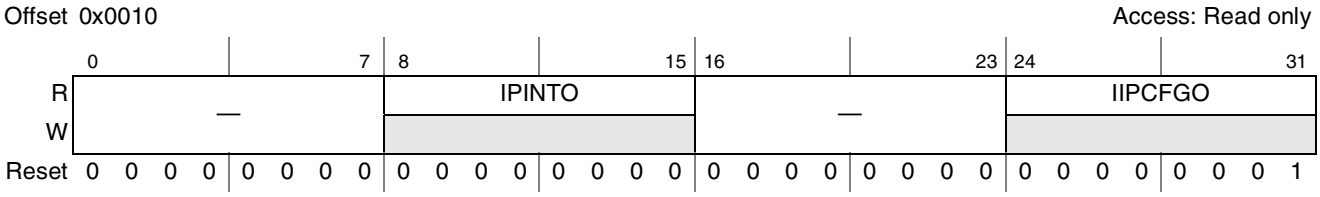


Figure 11-4. Block Revision Register 2 (BRR2)

Table 11-6 describes the BRR2 fields.

Table 11-6. BRR2 Field Descriptions

Bits	Name	Description
0–7	—	Reserved, should be cleared.
8–15	IPINTO	IP block integration options
16–23	—	Reserved, should be cleared.
24–31	IPCFGO	IP block configuration options

11.3.1.3 Feature Reporting Register (FRR)

FRR, shown in Figure 11-5, provides information about interrupt and processor core configurations. It also informs the programming environment of the controller version.

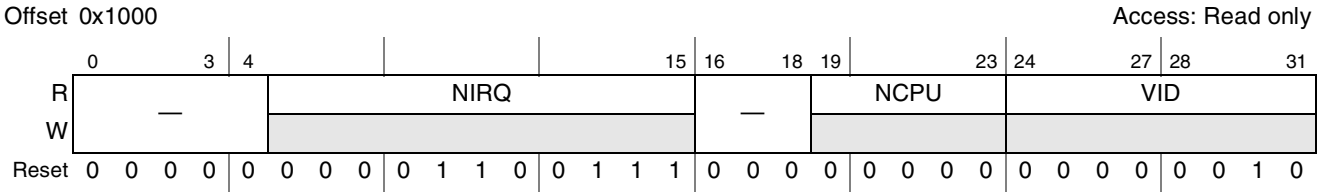


Figure 11-5. Feature Reporting Register (FRR)

Table 11-7 describes the FRR fields.

Table 11-7. FRR Field Descriptions

Bits	Name	Description
0–4	—	Reserved, should be cleared.
5–15	NIRQ	Number of interrupts. Holds the binary value of the number of the highest interrupt source supported minus one. The value is 103 (0x67 or 0b000_0110_0111). A zero in this field corresponds to one source.
16–18	—	Reserved, should be cleared
19–23	NCPU	Number of CPUs. The number of the highest physical CPUs (or processor cores) supported minus one. 00000 Single core—core0 00001 Two cores—core0 and core1
24–31	VID	Version ID. Reports the OpenPIC specification revision level supported by this interrupt controller implementation. The VID field's value of two (0x02) corresponds to revision 1.2 which is the revision level currently supported.

11.3.1.4 Global Configuration Register (GCR)

GCR, shown in Figure 11-6, controls the PIC's operating mode, and allows software to reset the PIC.

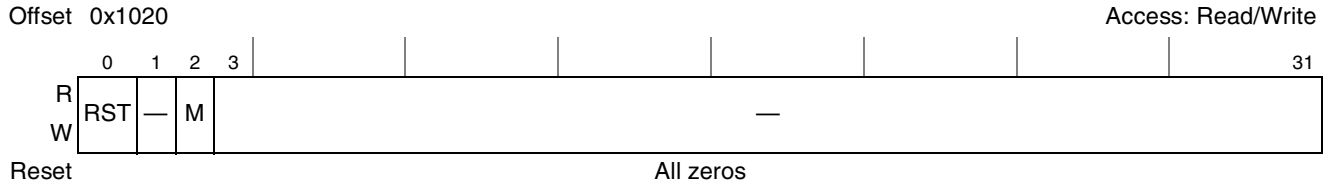


Figure 11-6. Global Configuration Register (GCR)

Table 11-8 describes the GCR fields.

Table 11-8. GCR Field Descriptions

Bits	Name	Description
0	RST	Reset. Setting RST forces the PIC to be reset. Cleared automatically when the reset sequence is complete. See Section 11.4.8, "Resetting the PIC," for more information.
1	—	Reserved, should be cleared.
2	M	Mode. PIC operating mode. Section 11.1.3, "Modes of Operation," provides details about these modes. 0 Pass-through mode. On-chip PIC is disabled and interrupts detected on IRQ0 are passed directly to core 0. 1 Mixed mode. Interrupts are handled by the normal priority and delivery mechanisms of the PIC.
3–31	—	Reserved, should be cleared.

11.3.1.5 Vendor Identification Register (VIR)

VIR, shown in Figure 11-7, is defined by the OpenPIC specifications and is provided for compliance. The zero value for VIR[VENDORID] indicates a generic OpenPIC-compliant device, which makes the other VIR fields meaningless.

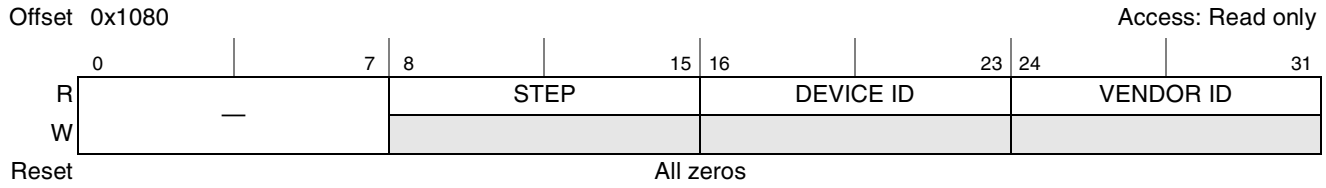


Figure 11-7. Vendor Identification Register (VIR)

Table 11-9 describes the VIR fields.

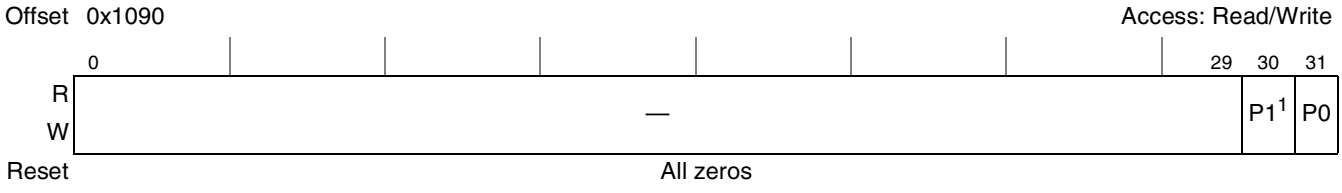
Table 11-9. VIR Field Descriptions

Bits	Name	Description
0–7	—	Reserved, should be cleared.
8–15	STEP	Stepping. Indicates the silicon revision for this device. Has no meaning if VENDOR ID value is zero.
16–23	DEVICE ID	Device identification. Vendor-specified identifier for this device. Has no meaning if VENDOR ID is zero.
24–31	VENDOR ID	Vendor identification. Specifies the manufacturer of this part. A value of zero implies a generic OpenPIC-compliant device.

11.3.1.6 Processor Core Initialization Register (PIR)

PIR, shown in Figure 11-8, provides a way for software to generate a core reset. Setting P1 or P0 causes the respective *sreset1* or *sreset0* signal to assert. These signals are routed to the appropriate *core_sreset* signals. Note that after requesting a core reset using this register the applicable bit should not be cleared until the requested core reset has occurred.

Note that although the OpenPIC architecture was defined to support up to 32 processing cores, only fields corresponding to the number of cores on the device are implemented.



[1] Reserved in single-processor implementations.

Figure 11-8. Processor Core Initialization Register (PIR)

Table 11-10 describes the PIR fields.

Table 11-10. PIR Field Descriptions

Bits	Name	Description
0–29	—	Reserved, should be cleared.
30	P1	Processor core 1 reset. Setting this bit causes the PIC to assert the *sreset1* signal. Reserved in single-processor implementations.
31	P0	Processor core 0 reset. Setting this bit causes the PIC to assert the *sreset0* signal.

11.3.1.7 Processor Reset Register (PRR)

The PRR provides a way for software on another device to perform the equivalent of a hardware reset (*hreset*) to the processor cores by writing to an PIC register. A *core0_hreset* and *core1_hreset* signal to the cores is asserted when a one is written to the corresponding processor reset field; these signals are held active until a zero is written to those same bits. This produces a pulse to the *core_hreset* input of the corresponding core or cores. Because of hardware-specific requirements on the width of such a pulse, to ensure successful hardware reset, the system must ensure a minimum adequate pulse width on *core_hreset*.

For proper system operation, a core should be reset in this way only if the core is already in nap or sleep state. Because a core in either state cannot perform the necessary write to cause a soft reset, a core cannot put itself into soft reset.

Note that although the OpenPIC architecture is defined to support up to 32 processing cores, only fields corresponding to processors on the device are implemented, as shown in Figure 11-9.

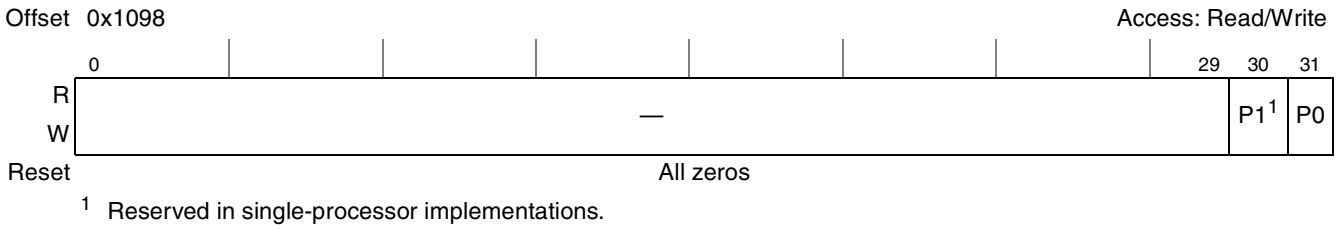


Figure 11-9. Processor Reset Register (PRR)

Table 11-11 describes the PRR fields.

Table 11-11. PRR Field Descriptions

Bits	Name	Description
0–29	—	Reserved, should be cleared.
30	P1	Processor 1 core reset. Setting P1 causes the PIC to assert the *core1_hreset* signal to processor 1. **Note:** Reserved in single-processor implementations.
31	P0	Processor 0 core reset. Setting P0 causes the PIC to assert the *core0_hreset*signal to processor 0.

11.3.1.8 Interprocessor Interrupt Vector/Priority Registers (IPIVPR0–IPIVPR3)

IPIVPRs, shown in Figure 11-10, contain the interrupt vector and priority fields for the four interprocessor interrupt channels. There is one vector/priority register per channel. The VECTOR and PRIORITY values

should not be changed while IPIVPR*n*[A] is set. See Section 11.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

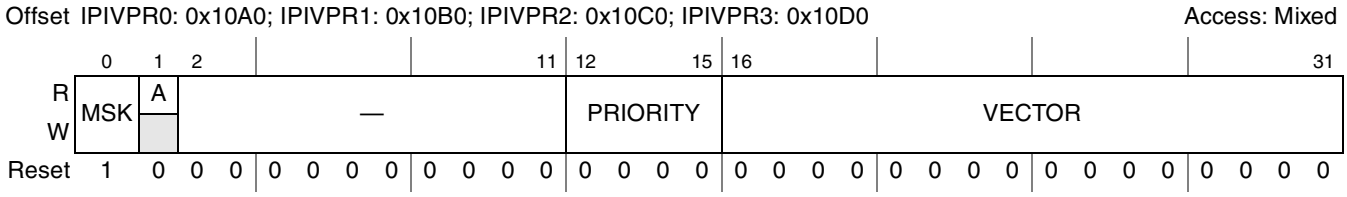


Figure 11-10. Interprocessor Interrupt Vector/Priority Register (IPIVPR*n*)

Table 11-12 describes the IPIVPR*n* fields.

Table 11-12. IPIVPR*n* Field Descriptions

Bits	Name	Description
0	MSK	Mask. Mask interrupts to *int* from this source. 0 An interrupt request is generated if the corresponding IPR bit is set. 1 Further interrupts from this source are disabled.
1	A	Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. 0 No current interrupt activity associated with this source. 1 The interrupt field for this source is set in the IPR or ISR.
2–11	—	Reserved, should be cleared.
12–15	PRIORITY	Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*.
16–31	VECTOR	Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 11-51.

11.3.1.9 Spurious Vector Register (SVR)

SVR, shown in Figure 11-11, contains the 16-bit vector returned to the processor core when the corresponding IACK register is read for a spurious interrupt.

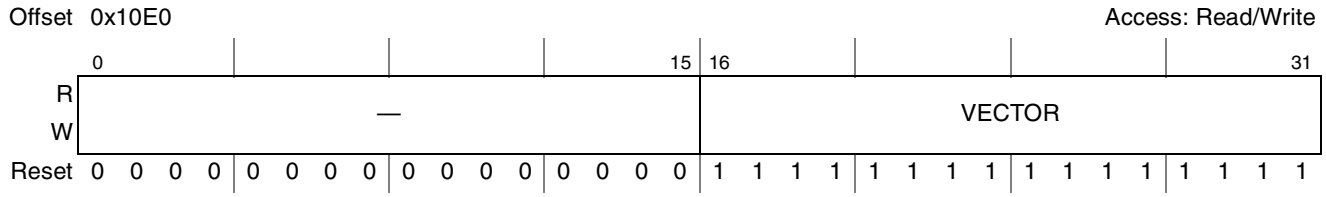


Figure 11-11. Spurious Vector Register (SVR)

Table 11-13 describes the SVR fields.

Table 11-13. SVR Field Descriptions

Bits	Name	Description
0–15	—	Reserved, should be cleared.
16–31	VECTOR	Spurious interrupt vector. Value returned when IACK is read during a spurious vector fetch. Section 11.4.1.2.3, "Spurious Vector Generation," gives information about the conditions that may cause a spurious vector fetch.

11.3.2 Global Timer Registers

The two independent groups of global timer registers, group A and group B, are identical in their functionality, except that they appear at different locations within the PIC register map. Note that each of the four timers within an *x* group have four individual configuration registers (GTCCR*xn*, GTBCR*xn*, GTVPR*xn*, GTDR*xn*), but they are only shown once in this section. These two groups of timers cannot be cascaded together.

11.3.2.1 Timer Frequency Reporting Register (TFRRA–TFRRB)

The TFRRs, shown in Figure 11-12, are written by software to report the clocking frequency of the PIC timers. Note that although TFRRs are read/write, the PIC ignores the register values.

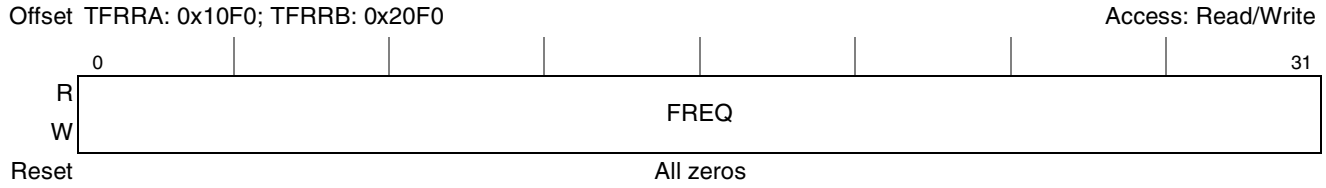


Figure 11-12. Timer Frequency Reporting Registers (TFRR*x*)

Table 11-14 describes the TFRR*x* registers.

Table 11-14. TFRR*x* Field Descriptions

Bits	Name	Description
0–31	FREQ	Timer frequency (in ticks/second (Hz)). Used to communicate the frequency of the global timers' clock source, (either the MPX clock or the frequency of the RTC signal), to user software. TFRR*x* is set only by software for later use by other applications and its value in no way affects the operating frequency of the global timers. The timers operate at a ratio of this clock frequency, as set by TCR*x*[CLKR]. See Section 11.3.2.6, "Timer Control Registers (TCRA–TCRB)."

11.3.2.2 Global Timer Current Count Registers (GTCCRA0–GTCCRA3, GTCCRB0–GTCCRB3)

The GTCCRs, shown in Figure 11-13, contain the current count for each of the four PIC timers in each of the two groups.

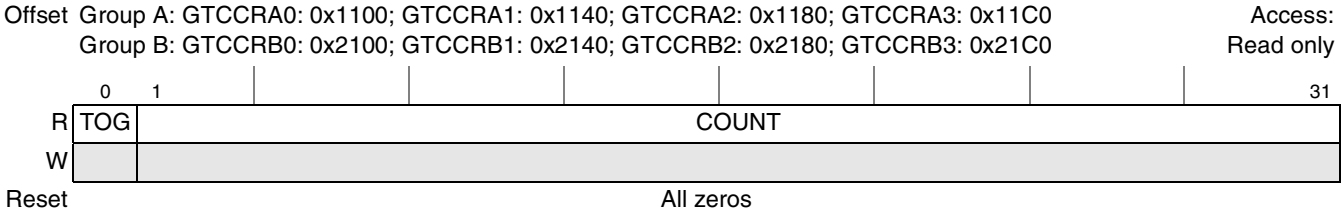


Figure 11-13. Global Timer Current Count Registers (GTCCR*xn*)

Table 11-15 describes the GTCCR*xn* fields.

Table 11-15. GTCCR*xn* Field Descriptions

Bits	Name	Description
0	TOG	Toggle. Toggles when the current count decrements to zero. Cleared when GTBCR*xn*[CI] goes from 1 to 0.
1–31	COUNT	Current count. Decremented while GTBCR*xn*[CI] is zero. When the timer count reaches zero, an interrupt is generated (provided it is not masked), the toggle bit is inverted, and the count is reloaded. For non-cascaded timers, the reload value is the contents of the corresponding GTBCR*xn*. Cascaded timers are reloaded with either all ones, or the GTBCR*xn* contents, depending on the value of TCR*n*[ROVR]. See Section 11.3.2.6, “Timer Control Registers (TCRA–TCRB),” for more details.

11.3.2.3 Global Timer Base Count Registers (GTBCRA0–GTBCRA3, GTBCRB0–GTBCRB3)

The GTBCRs contain the base counts for each of the four PIC timers in each of the two groups, as shown in Figure 11-14. This value is reloaded into the corresponding GTCCR*xn* when the current count reaches zero. Note that when zero is written to the base count field, (and GTCCR*xn*[CI] = 0), the timer generates an interrupt on every timer cycle.

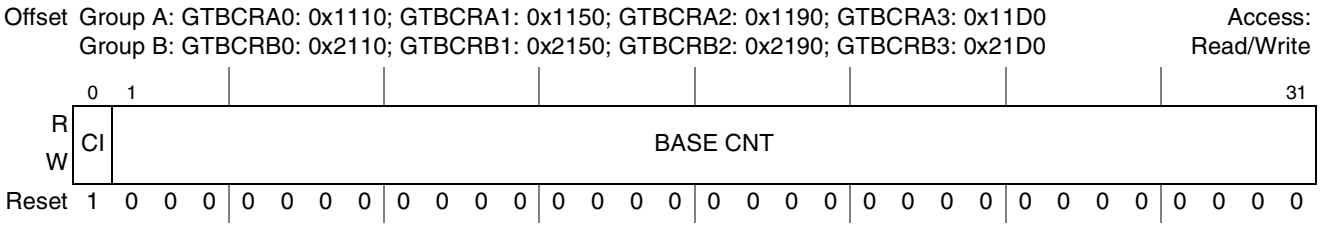


Figure 11-14. Global Timer Base Count Register (GTBCR*xn*)

Table 11-16 describes the GTBCR*xn* fields.

Table 11-16. GTBCR*xn* Field Descriptions

Bits	Name	Description
0	CI	Count inhibit. Always set following reset 0 Counting enabled 1 Counting inhibited
1–31	BASE CNT	Base count. When CI transitions from 1 to 0, this value is copied into the corresponding GTCCR*xn* and the toggle bit is cleared. If CI is already cleared (counting is in progress), the base count is copied to the GTCCR*xn* at the next zero crossing of the current count.

11.3.2.4 Global Timer Vector/Priority Registers (GTVPRA0–GTVPRA3, GTVPRB0–GTVPRB3)

The GTVPRs contain the interrupt vector and the interrupt priority values for the timers as shown in Figure 11-15. They also contain the mask and activity fields for all the timers. See Section 11.4.1, "Flow of Interrupt Control," for information on IPR and ISR.

Offset Group A: GTVPRA0: 0x1120; GTVPRA1: 0x1160; GTVPRA2: 0x11A0; GTVPRA3: 0x11E0; Access: Mixed
Group B: GTVPRB0: 0x2120; GTVPRB1: 0x2160; GTVPRB2: 0x21A0; GTVPRB3: 0x21E0

	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
R W	MSK	A	—										PRIORITY				VECTOR															
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	0	0	0	0

Figure 11-15. Global Timer Vector/Priority Register (GTVPR*xn*)

Table 11-17 describes the GTVPR*xn* fields.

Table 11-17. GTVPR*xn* Field Descriptions

Bits	Name	Description
0	MSK	Mask. Mask interrupts to *int* from this source. 0 An interrupt request is generated if the corresponding IPR bit is set. 1 Further interrupts from this source are disabled.
1	A	Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. 0 No current interrupt activity associated with this source. 1 The interrupt field for this source is set in the IPR or ISR.
2–11	—	Reserved, should be cleared.
12–15	PRIORITY	Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*.
16–31	VECTOR	Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 11-51.

11.3.2.5 Global Timer Destination Registers (GTDRA0–GTDRA3, GTDRB0–GTDRB3)

The GTDR*xn* registers, shown in Figure 11-16, control the destination (core) to which each timer's interrupt is directed. Note that GTDR*xn* bits can be set independently of each other and that either P1 or P0 or both can be set for this type of interrupt.

Offset Group A: GDTRA0: 0x1130; GDTRA1: 0x1170; GDTRA2: 0x11B0; GDTRA3: 0x11F0; Access:
Group B: GDTRB0: 0x2130; GDTRB1: 0x2170; GDTRB2: 0x21B0; GDTRB3: 0x21F0 Read/Write

	0																								29	30	31
R W	—																									P1[1]	P0
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] Reserved in single-processor implementations.

Figure 11-16. Global Timer Destination Registers (GTDR*xn*)

Table 11-18 describes the GTDR*xn* fields.

Table 11-18. GTDR*xn* Field Descriptions

Bits	Name	Description
0–29	—	Reserved, should be cleared.
30	P1	Processor core 1. This interrupt is multicasting, so both P0 and P1 can be set. 0 Processor core 1 does not receive this interrupt 1 Directs the timer interrupt to processor core 1 **Note:** Reserved in single-processor implementations.
31	P0	Processor core 0. Default destination after PIC is reset. Both P0 and P1 can be set. 0 Processor core 0 does not receive this interrupt. 1 Directs the timer interrupt to processor core 0.

11.3.2.6 Timer Control Registers (TCRA–TCRB)

The TCR registers, shown in Figure 11-18, provide various configuration options such as count frequency and roll-over behavior for the timers.

There are two choices for the clock source for the timers: a selectable frequency ratio from the MPX bus clock, or the RTC signal. TCRs can be cascaded to create timers larger than the default 31-bit global timers. Timer cascade fields allow configuration of up to two 63-bit timers, one 95-bit timer, or one 127-bit timer (within each group).

With one exception mentioned below, the value reloaded into a timer is determined by its roll-over control field, TCR*x*[ROVR]. Setting TCR*x*[ROVR] causes its GTCCR*xn* to roll over to all ones when the count reaches zero. This is equivalent to reloading the count register with 0xFFFF_FFFF instead of its base count value. Clearing a timer's associated ROVR bit ensures the timer always reloads with its base count value.

When timers are cascaded, the last (most significant) counter in the cascade also affects their roll-over behavior. Cascaded timers always reload their base count when the most significant counter has decremented to zero, regardless of the TCR*x*[ROVR] settings.

For example, timers 0–2 can be cascaded to generate one interrupt per hour. As shown in Table 11-19, given an MPX clock frequency of 333 MHz, letting the timer clock frequency default to 1/8th the system clock, (TCR*x*[CLKR] = 0 sets a clock ratio of 8), provides a basic input of 41.625 MHz to timer 0. Setting timer 0 to count 41,625,000 (0x27B_25A8) timer clock cycles generates one output per second. Setting both timers 1 and 2 to 59, and cascading all three timers, generates one interrupt every hour from timer 2.

Table 11-19. Parameters for Hourly Interrupt Timer Cascade Example

System Clock	Clock Ratio	Timer Clock	Timer 0 Count	Timer 1 Count	Timer 2 Count
333 MHz	1 / 8	41.625 MHz	41.625 x 10^6 (0x027B_25A8)	59[1] (0x0000_0036)	59 (0x0000_0036)

[1] Counting down from 59 through 0 requires 60 ticks.

$$(41.625 \text{ x } 106 \text{ ticks/sec}) \times (60 \text{ sec/min}) \times (60 \text{ min/hr}) = \text{total ticks/hr generating 1 interrupt/hr}$$

Figure 11-17. Example Calculation for Cascaded Timers

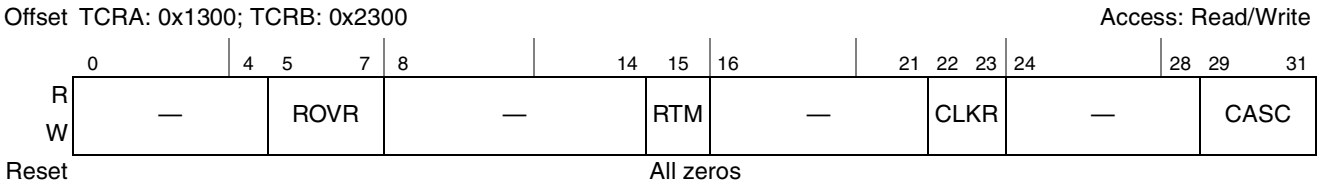


Figure 11-18. Timer Control Registers (TCR*x*)

Table 11-20 describes the TCR*x* fields.

Table 11-20. TCR*x* Field Descriptions

Bits	Name	Description
0–4	—	Reserved, should be cleared.
5–7	ROVR	Roll-over control for cascaded timers only. Specifies behavior when count reaches zero by identifying the source of the reload value. Cascaded timers are always reloaded with their base count value when the more significant timer in the cascade (the upstream timer) is zero. Bits 5–7 correspond to timers 2–0. Note that global timer 3 always reloads with its GTBCR*xn*. 0 The timer does not roll over. When the count reaches zero, GTCCR*xn* is reloaded with the GTBCR*xn* value. 1 Timer rolls over at zero to all ones. (When the count reaches zero, GTCCR*xn* is reloaded with 0xFFFF_FFFF.) 000 All timers reload with base count. 001 Timers 1 and 2 reload with base count, timer 0 rolls over (reloads with 0xFFFF_FFFF). 010 Timers 0 and 2 reload with base count, timer 1 rolls over (reloads with 0xFFFF_FFFF). 011 Timer 2 reloads with base count, timers 0 and 1 roll over (reload with 0xFFFF_FFFF). 100 Timers 0 and 1 reload with base count, timer 2 rolls over (reloads with 0xFFFF_FFFF). 101 Timer 1 reloads with base count, timers 0 and 2 roll over (reload with 0xFFFF_FFFF). 110 Timer 0 reloads with base count, timers 1 and 2 roll over (reload with 0xFFFF_FFFF). 111 Timers 0, 1, and 2 roll over (reload with 0xFFFF_FFFF).
8–14	—	Reserved, should be cleared.
15	RTM	Real time mode. Specifies the clock source for the PIC timers. 0 Timer clock frequency is a ratio of the frequency of the MPX clock as determined by the CLKR field. This is the default value. 1 The RTC signal is used to clock the PIC timers. If this bit is set, the CLKR field has no meaning.

Table 11-20. TCR*x* Field Descriptions (continued)

Bits	Name	Description
16–21	—	Reserved, should be cleared.
22–23	CLKR	Clock ratio. Specifies the ratio of the timer frequency to the MPX clock. The following are supported: 00 Default. Divide by 8 01 Divide by 16 10 Divide by 32 11 Divide by 64
24–28	—	Reserved, should be cleared.
29–31	CASC	Cascade timers. Specifies the output of particular global timers as input to others. 000 Default. Timers not cascaded 001 Cascade timers 0 and 1 010 Cascade timers 1 and 2 011 Cascade timers 0, 1, and 2 100 Cascade timers 2 and 3 101 Cascade timers 0 and 1; timers 2 and 3 110 Cascade timers 1, 2, and 3 111 Cascade timers 0, 1, 2, and 3

11.3.3 IRQ_OUT and Critical Interrupt Summary Registers

The summary registers indicate the specific interrupt sources routed to the $\overline{\text{IRQ\_OUT}}$ or *cint0/cint1*. PIC outputs. Summary register bits are cleared when the corresponding interrupt that caused a bit to be set is negated. Note that only level-sensitive interrupts can be routed to $\overline{\text{IRQ\_OUT}}$ or *cint0* and *cint1*.

The IRQ_OUT summary registers, shown in Figure 11-20 through Figure 11-22 contain one bit for each interrupt source that can be routed to $\overline{\text{IRQ\_OUT}}$. The corresponding bit is set if the interrupt is active and is routed to $\overline{\text{IRQ\_OUT}}$ (that is, if the corresponding *x*IDR*n*[EP] is set).

The critical interrupt summary registers, shown in Figure 11-23 through Figure 11-25, contain one bit for each interrupt source that can be designated as a critical interrupt. The corresponding bit is set if the interrupt is active and is routed to either the *cint* outputs of the PIC (if *x*IDR*n*[CI*n*] = 1 in its corresponding destination register).

11.3.3.1 External Interrupt Summary Register (ERQSR)

NOTE

ERQSR fields report only the current logic level of IRQ0–IRQ11 pins. These fields were designed to work with level-sensitive interrupts; values returned for edge-sensitive interrupts may be unreliable.

Figure 11-19 shows the ERQSR fields.

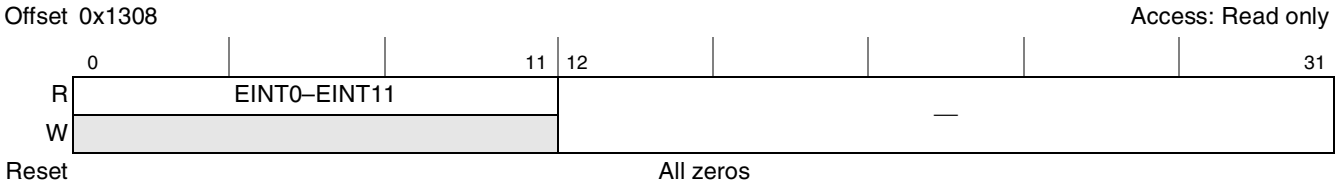


Figure 11-19. External Interrupt Summary Register (ERQSR)

Table 11-21 describes the ERQSR fields.

Table 11-21. ERQSR Field Descriptions

Bits	Name	Description
0–11	EINT*n*	External interrupts signal 0–11 status. Bit 0 represents EINT0. Bit 11 represents EINT 11. 0 The corresponding external interrupt signal is at a low logic level. 1 The corresponding external interrupt signal is at a high logic level.
12–31	—	Reserved, should be cleared.

11.3.3.2 IRQ_OUT Summary Register 0 (IRQSR0)

Figure 11-20 shows the IRQSR0 fields.

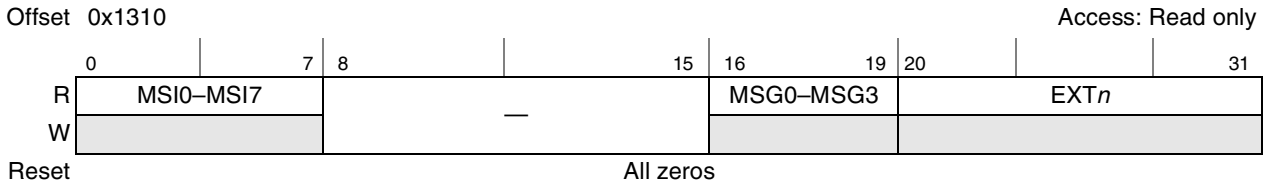


Figure 11-20. IRQ_OUT Summary Register 0 (IRQSR0)

Table 11-22 describes the IRQSR0 fields.

Table 11-22. IRQSR0 Field Descriptions

Bits	Name	Description
0–7	MSI*n*	Shared message signaled interrupt *n* status 0 Interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$. 1 Interrupt is active and is routed to the $\overline{\text{IRQ\_OUT}}$ signal (that is, if the corresponding *x*IDR*n*[EP] is set).
8–15	—	Reserved, should be cleared.

Table 11-22. IRQSR0 Field Descriptions (continued)

Bits	Name	Description
16–19	MSG*n*	Message interrupt *n* status 0 Interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$. 1 Interrupt is active and is routed to the $\overline{\text{IRQ\_OUT}}$ signal (that is, if the corresponding *x*IDR*n*[EP] is set).
20–31	EXT*n*	External interrupts 0–11. Each bit corresponds to a unique interrupt according to the following: Bit Interrupt 20 IRQ0 21 IRQ1 … 31 IRQ11 0 The corresponding interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$. 1 The corresponding interrupt is active and routed to $\overline{\text{IRQ\_OUT}}$ (if the corresponding *x*IDR*n*[EP] is set).

11.3.3.3 IRQ_OUT Summary Register 1 (IRQSR1)

Figure 11-21 shows the IRQSR1 fields.

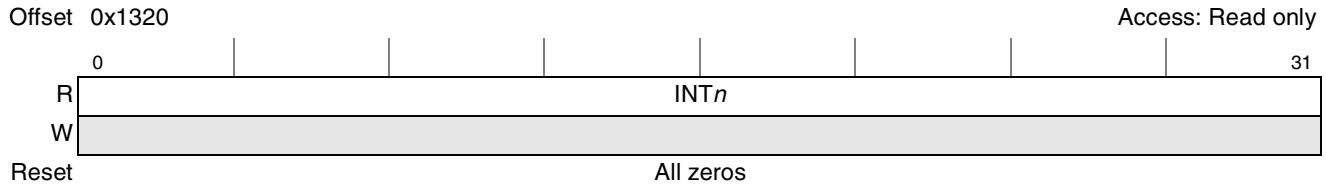


Figure 11-21. IRQ_OUT Summary Register 1 (IRQSR1)

Table 11-23 describes the IRQSR1 fields.

Table 11-23. IRQSR1 Field Descriptions

Bits	Name	Description
0–31	INT*n*	Internal interrupts 0–31 status. Bit 0 represents INT0. Bit 31 represents INT31. 0 The corresponding interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$. 1 The corresponding interrupt is active and is routed to $\overline{\text{IRQ\_OUT}}$ (that is, if the corresponding *x*IDR*n*[EP] is set).

11.3.3.4 IRQ_OUT Summary Register 2 (IRQSR2)

Figure 11-22 shows the IRQSR2 fields.

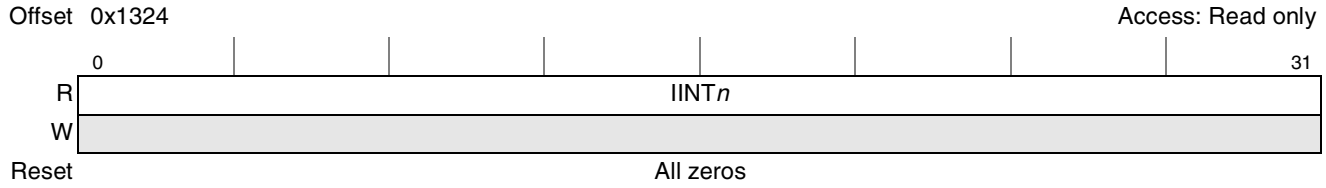


Figure 11-22. IRQ_OUT Summary Register 2 (IRQSR2)

Table 11-24 describes the IRQSR2 fields.

Table 11-24. IRQSR2 Field Descriptions

Bits	Name	Description
0–15	INT*n*	Internal interrupts 32–47 status. Bit 0 represents INT32. Bit 15 represents INT47. 0 The corresponding interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$. 1 The corresponding interrupt is active and is routed to $\overline{\text{IRQ\_OUT}}$, if the corresponding *x*IDR*n*[EP] is set.
16–31	—	Reserved, should be cleared.
0–31	INT*n*	Internal interrupts 32–63 status. Bit 0 represents INT32. Bit 31 represents INT63. 0 The corresponding interrupt is not active or not routed to $\overline{\text{IRQ\_OUT}}$. 1 The corresponding interrupt is active and is routed to $\overline{\text{IRQ\_OUT}}$, if the corresponding *x*IDR*n*[EP] is set.

11.3.3.5 Critical Interrupt Summary Register 0 (CISR0)

Figure 11-23 shows CISR0.

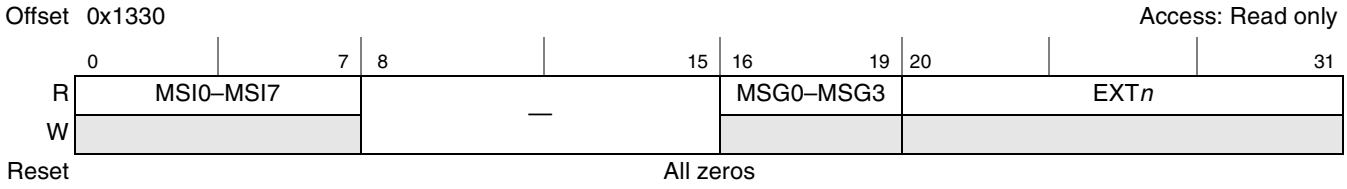


Figure 11-23. Critical Interrupt Summary Register 0 (CISR0)

Table 11-25 describes CISR0 fields.

Table 11-25. CISR0 Field Descriptions

Bits	Name	Description
0–7	MSI*n*	Shared message signaled interrupts 0–7. Bit 0 represents MSI0; bit 7 represents MSI7. 0 The corresponding interrupt is not active or not routed to *cint*. 1 The corresponding interrupt is active and is routed to *cint*, if the corresponding *x*IDR*n*[CI] is set.
8–15	—	Reserved, should be cleared.
16–19	MSG*n*	Message interrupts 0–3. Bit 16 represents MSG0; bit 19 represents MSG3. 0 The corresponding interrupt is not active or not routed to *cint*. 1 The corresponding interrupt is active and is routed to *cint* (if the corresponding *x*IDR*n*[CI] is set).
20–31	EXT*n*	External interrupts 0–11. Bit 20 represents IRQ0. Bit 31 represents IRQ11. 0 The corresponding interrupt is not active or not routed to *cint*. 1 The corresponding interrupt is active and is routed to *cint* (if the corresponding *x*IDR*n*[CI] is set).

11.3.3.6 Critical Interrupt Summary Register 1 (CISR1)

Figure 11-24 shows the CISR1.

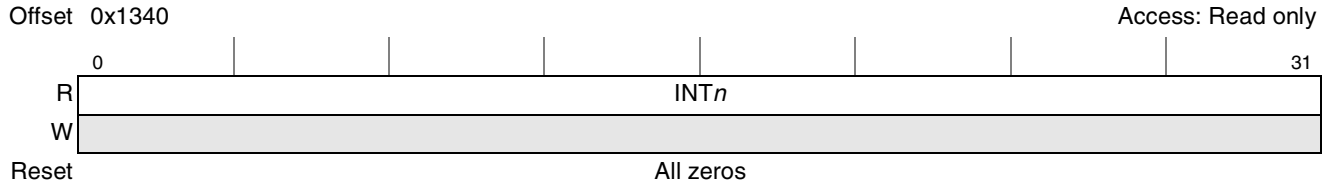


Figure 11-24. Critical Interrupt Summary Register 1 (CISR1)

Table 11-26 describes CISR1.

Table 11-26. CISR1 Field Descriptions

Bits	Name	Description
0–31	INT*n*	Internal interrupts 0–31. Bit 0 represents INT0. Bit 31 represents INT31. 0 Corresponding interrupt is not active or not routed to *cint*. 1 The corresponding interrupt is active and is routed to the *cint* (if the corresponding *x*IDR*n*[CI] is set).

11.3.3.7 Critical Interrupt Summary Register 2 (CISR2)

Figure 11-25 shows the CISR2.

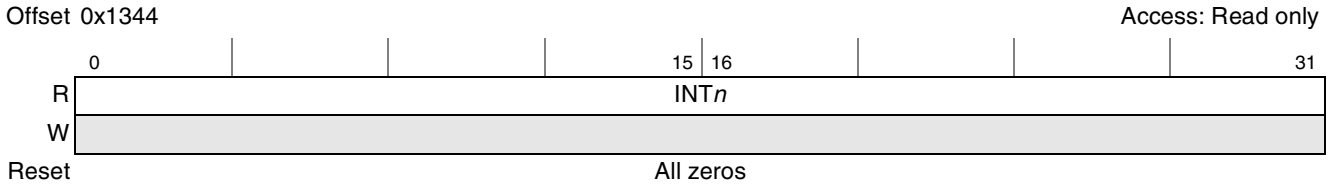


Figure 11-25. Critical Interrupt Summary Register 2 (CISR2)

Table 11-27 describes CISR2.

Table 11-27. CISR2 Field Descriptions

Bits	Name	Description
0–31	INT*n*	Internal interrupts 32–63. Bit 0 represents INT32. Bit 31 represents INT63. 0 Corresponding interrupt is not active or not routed to *cint*. 1 The corresponding interrupt is active and is routed to the *cint*, if the corresponding *x*IDR*n*[CI] is set.

11.3.4 Performance Monitor Mask Registers (PMMRs)

The twelve performance monitor mask registers consist of four sets of three 32-bit registers, PM*n*MR0, PM*n*MR1, and PM*n*MR2. Each set can be configured to select one interrupt source (interprocessor, timer, message, shared message signaled, external, or internal) to generate a performance monitor event. The performance monitor can be configured to track this event in the performance monitor local control registers. See Section 24.3.1.2, “Performance Monitor Local Control Registers (PMLCAn, PMLCBn).”

11.3.4.1 Performance Monitor Mask Registers 0 (PM0MR0–PM3MR0)

Each PM*n*MR0 register, shown in Figure 11-26, is matched with a PM*n*MR1 and a PM*n*MR2 register. Because each unreserved bit in the 96-bit vector (PM*n*MR0/1/2) specifies a different interrupt, only one bit in the 96-bit vector can be unmasked at a time. Unmasking more than one bit per set is considered a programming error and results in unpredictable behavior.

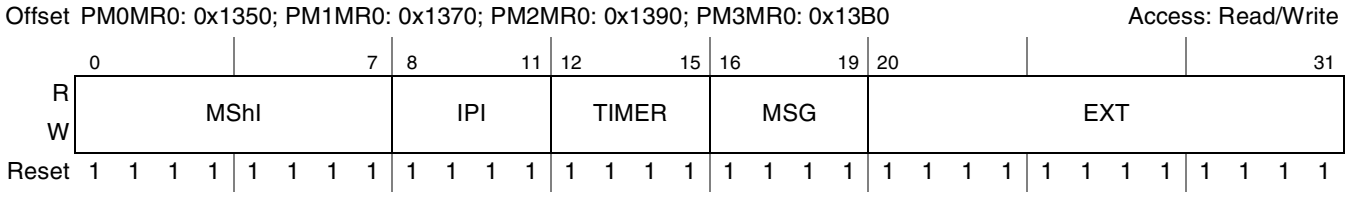


Figure 11-26. Performance Monitor Mask Registers 0 (PM*n*MR0)

Table 11-28 describes the PM*n*MR0 fields.

Table 11-28. PM*n*MR0 Field Descriptions

Bits	Name	Description
0–7	MShI	Shared message signaled interrupts 0–7 0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs. 1 The corresponding interrupt does not generate a performance monitor event.
8–11	IPI	Interprocessor interrupts 0–3 0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs. 1 The corresponding interrupt does not generate a performance monitor event.
12–15	TIMER	Timer interrupts 0–3 (Group A and Group B: Each bit represents an OR of the event for the correspondingly numbered timer in Group A and that in Group B). 0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs. 1 The corresponding interrupt does not generate a performance monitor event.
16–19	MSG	Message interrupts 0–3 0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs. 1 The corresponding interrupt does not generate a performance monitor event.
20–31	EXT	External interrupts IRQ[0:11] 0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs. 1 The corresponding interrupt does not generate a performance monitor event.

11.3.4.2 Performance Monitor Mask Registers 1 (PM0MR1–PM3MR1)

Figure 11-27 shows the PM*n*MR1 registers.

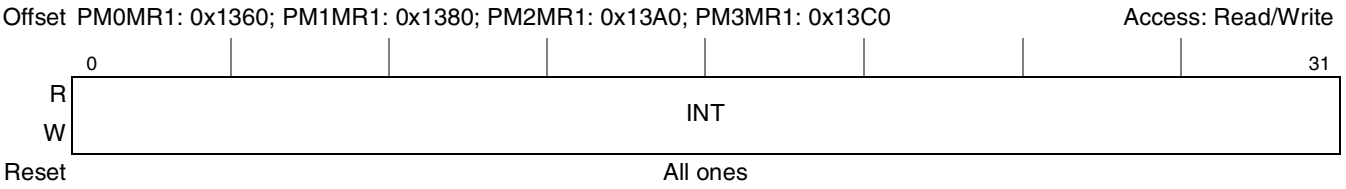


Figure 11-27. Performance Monitor Mask Registers 1 (PM*n*MR1)

Table 11-29 describes the PM*n*MR1 registers.

Table 11-29. PM*n*MR1 Field Descriptions

Bits	Name	Description
0–31	INT	Internal interrupts 0–31 0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs. 1 The corresponding interrupt does not generate a performance monitor event.

11.3.4.3 Performance Monitor Mask Registers 2 (PM0MR2–PM3MR2)

Figure 11-28 shows the PM*n*MR2 registers.

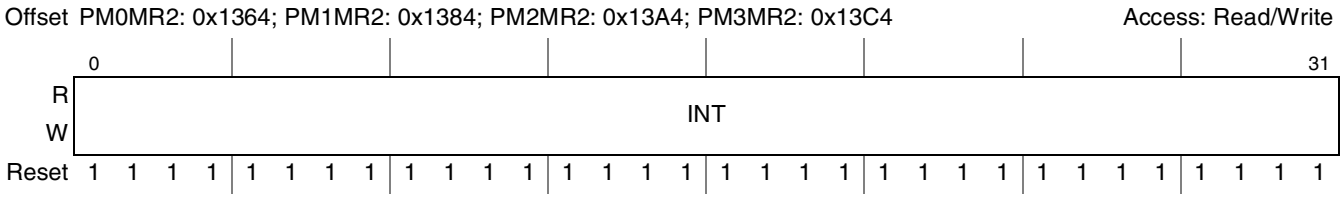


Figure 11-28. Performance Monitor Mask Registers 2 (PM*n*MR2)

Table 11-30 describes the PM*n*MR2 registers.

Table 11-30. PM*n*MR2 Field Descriptions

Bits	Name	Description
0–31	INT	Internal interrupts 32–64 0 The corresponding interrupt source generates a performance monitor event when the interrupt occurs. 1 The corresponding interrupt does not generate a performance monitor event.

11.3.5 Message Registers

The following registers support the message register interrupts:

- Section 11.3.5.1, “Message Registers (MSGR0–MSGR3)”
- Section 11.3.5.2, “Message Enable Register (MER)”
- Section 11.3.5.3, “Message Status Register (MSR)”
- Section 11.3.7.5, “Messaging Interrupt Vector/Priority Registers (MIVPRn)”
- Section 11.3.7.6, “Messaging Interrupt Destination Registers (MIDR0–MIDR3)”

Writing to one of the four message registers (MSGR0–MSGR3) causes a messaging interrupt as directed by the other message registers listed above. Reading a message register clears the messaging interrupt. Note that a messaging interrupt can also be cleared by writing a one to the corresponding status field of the PIC message status register (MSR), shown in Figure 11-31.

11.3.5.1 Message Registers (MSGR0–MSGR3)

The message registers (MSGR0–MSGR3), shown in Figure 11-29, can contain a 32-bit message.

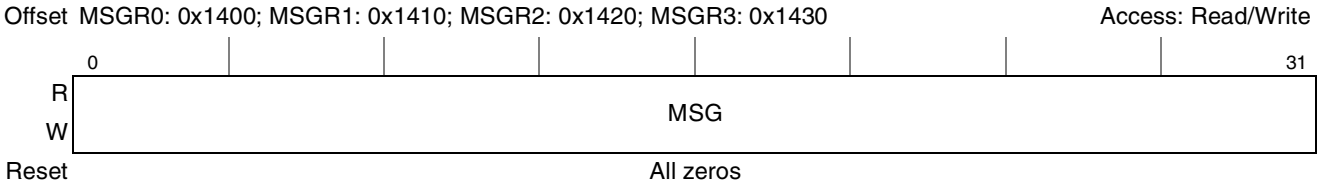


Figure 11-29. Message Registers (MSGRs)

Table 11-31 describes the MSGR registers.

Table 11-31. MSGR*n* Field Descriptions

Bits	Name	Description
0–31	MSG	Message. Contains the 32-bit message data.

11.3.5.2 Message Enable Register (MER)

The MER, shown in Figure 11-30, contains the enable bits for each message register. The enable bit must be set to enable interrupt generation when the corresponding message register is written.

When bits in MER are set to mask message interrupts, an interrupt is not generated if the message register is written while it is masked in MER and the MER bit is then cleared. To mask the interrupt without loss, set MIVPR*n*[MSK]. (See Section 11.3.7.5, “Messaging Interrupt Vector/Priority Registers (MIVPRn).”) MER should be set to 0x0000_000F at reset and then left unchanged during normal operation.

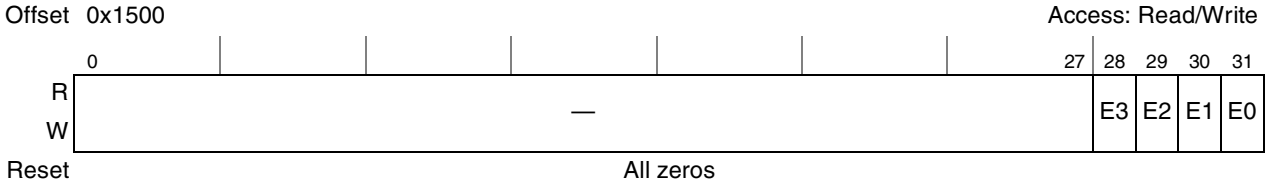


Figure 11-30. Message Enable Register (MER)

Table 11-32 describes the MER fields.

Table 11-32. MER Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared.
28–32	E*n*	Enable 3–enable 0. Used to enable interrupt generation for MSGR*n* (where *n* = 0–3). 0 Interrupt generation for MSGR*n* disabled. 1 Interrupt generation for MSGR*n* enabled.

11.3.5.3 Message Status Register (MSR)

The message status register (MSR) shown in Figure 11-31 contains status bits for each message register. A status bit is set when the corresponding messaging interrupt is active. Writing a 1 to a status bit clears the corresponding message interrupt and the status bit.

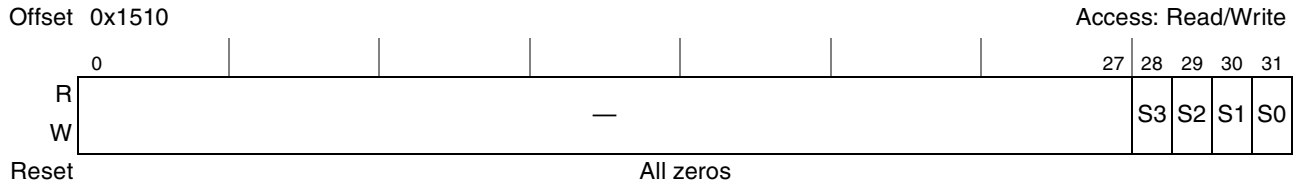


Figure 11-31. Message Status Register (MSR)

Table 11-33 describes the MSR fields.

Table 11-33. MSR Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared.
28	S*n*	Status 3–status 0. Reports status of messaging interrupt *n*. Writing a 1 clears this field. 0 Messaging interrupt *n* is not active. 1 Messaging interrupt *n* is active.

11.3.6 Shared Message Signaled Registers

This section contains description the shared message signaled interrupt registers (MSIRs). The shared message signaled interrupt structure allows programs to interrupt each other by simply writing to these shared memory-mapped registers in the PIC. Each of the eight MSIRs can be thought of as collecting interrupts from 32 different memory-mapped writes that can cause interrupts.

11.3.6.1 Shared Message Signaled Interrupt Registers (MSIR0–MSIR7)

The eight MSIRs indicate which of the up to 32 interrupt sources sharing the message register have pending interrupts. These registers are cleared when read. A write to these registers has no effect.

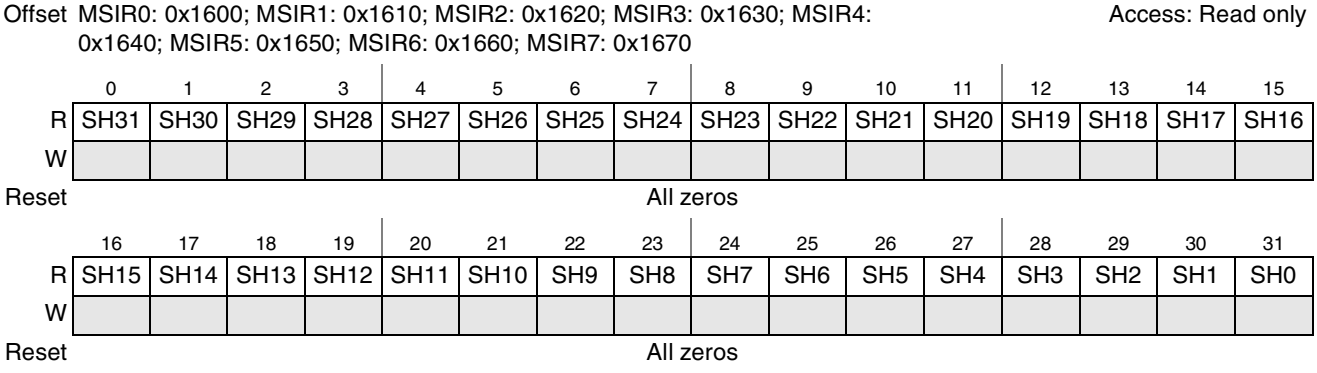


Figure 11-32. Message Signaled Interrupt Registers (MSIR*n*)

Table 11-34 describes the bits of the MSIRs.

Table 11-34. MSIR*n* Field Descriptions

Bits	Name	Description
n	SH*n*	Message sharer *n* has a pending interrupt.

11.3.6.2 Shared Message Signaled Interrupt Status Register (MSISR)

MSISR, shown in Figure 11-33, contains the status bits for the shared message signaled interrupts. A status bit is set when the corresponding MSIR has an active interrupt. The status bit is 0 if all the corresponding shared interrupt sources are cleared for that MSIR.

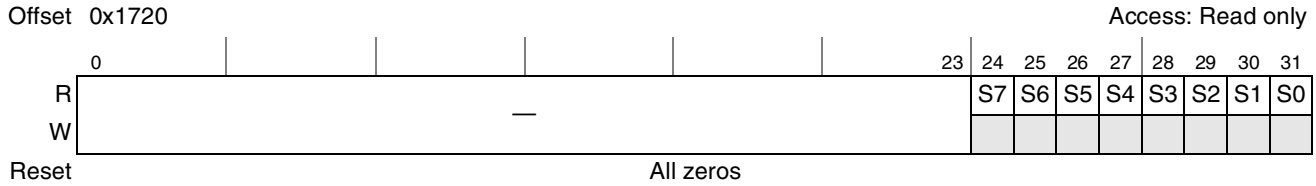


Figure 11-33. Shared Message Signaled Interrupt Status Register (MSISR)

Table 11-35 describes the bits of the MSISR.

Table 11-35. MSISR Field Descriptions

Bits	Name	Description
0–23	—	Reserved, should be cleared.
24–31	S*n*	Status *n*. 0 MSIR*n* is not active. 1 MSIR*n* has an active interrupt.

11.3.6.3 Shared Message Signaled Interrupt Index Register (MSIIR)

MSIIR, shown in Figure 11-33, provides the mechanism for setting an interrupt in the MSIRs. When MSIIR is written, MSIIR[SRS] selects the register in which an interrupt bit is to be set; MSIIR[IBS] selects the shared interrupt field in the selected MSIR register to be set. MSIIR is primarily intended to support PCI Express MSIs.

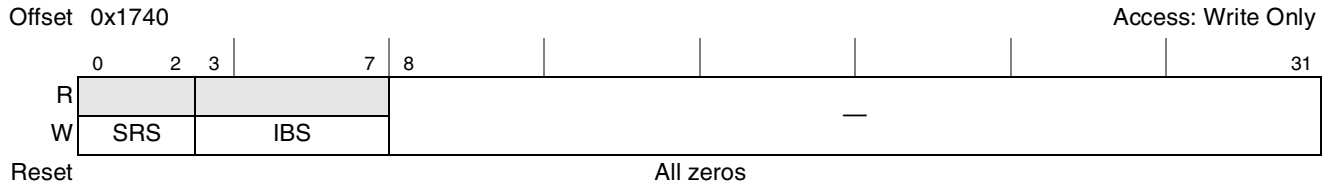


Figure 11-34. Shared Message Signaled Interrupt Index Register (MSIIR)

Table 11-36 describes the bits of the MSIIR.

Table 11-36. MSIIR Field Descriptions

Bits	Name	Description
0–2	SRS	Shared interrupt register select. Selects the MSIR to be written 000 MSIR 0 001 MSIR 1 010 MSIR 2 ... 111 MSIR 7
3–7	IBS	Interrupt bit select—Selects the bit to set in the MSIR 00000Set field SH0 (bit 31) 00001Set field SH1 (bit 30) 00010Set field SH2 (bit 29) ... 11111Set field SH31 (bit 0)
8–31	—	Reserved, should be cleared.

11.3.6.4 Shared Message Signaled Interrupt Vector/Priority Register (MSIVPRs)

The MSIVPRs, shown in Figure 11-35, have the same fields and format as the GTVPRs. See Section 11.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

Offset MSIVPR0: 0x1C00; MSIVPR1: 0x1C20; MSIVPR2: 0x1C40; MSIVPR3: 0x1C60; MSIVPR4: 0x1C80; MSIVPR5: 0x1CA0; MSIVPR6: 0x1CC0; MSIVPR7: 0x1CE0 — Access: Mixed

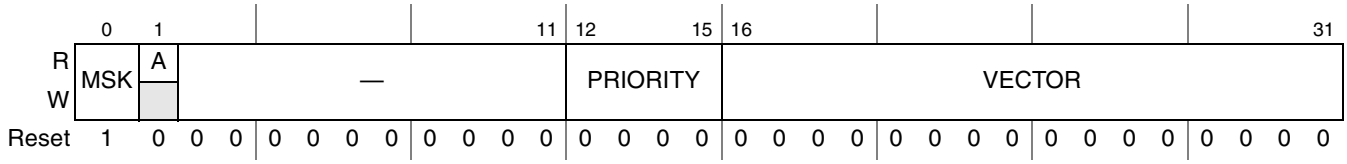


Figure 11-35. Shared Message Signaled Interrupt Vector/Priority Register (MSIVPRs)

Table 11-37 describes the bits of the MSIVPRs.

Table 11-37. MSIVPRn Field Descriptions

Bits	Name	Description
0	MSK	Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*. 0 An interrupt request is generated if the corresponding IPR bit is set. 1 Further interrupts from this source are disabled.
1	A	Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*. 0 No current interrupt activity associated with this source. 1 The interrupt field for this source is set in the IPR or ISR.
2–11	—	Reserved, should be cleared.
12–15	PRIORITY	Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*.
16–31	VECTOR	Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 11-50.

11.3.6.5 Shared Message Signaled Interrupt Destination Registers 0–7 (MSIDR*n*)

The MSIDRs, shown in Figure 11-36, contain the destination fields for shared message signaled interrupts. Only one destination bit may be set; otherwise, behavior is undefined.

Offset MSIDR0: 0x1C10; MSIDR1: 0x1C30; MSIDR2: 0x1C50; MSIDR3: 0x1C70; MSIDR4: 0x1C90; MSIDR5: 0x1CB0; MSIDR6: 0x1CD0; MSIDR7: 0x1CF0 — Access: Read/Write

	0	1	2																								29	30	31
R W	EP	CI0	CI1[1]	—																								P1[1]	P0
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] Reserved in single-processor implementations.

Figure 11-36. Shared Message Signaled Interrupt Destination Registers (MSIDR*n*)

Table 11-38 describes MSIDR*n* fields.

Table 11-38. MSIDR*n* Field Descriptions

Bits	Name	Description
0	EP	External signal. Allows interrupt to be serviced externally. 0 Interrupt is not routed to $\overline{\text{IRQ\_OUT}}$. 1 Interrupt is routed to $\overline{\text{IRQ\_OUT}}$ for external servicing.
1	CI0	Critical interrupt 0. 0 Processor core 0 does not receive this interrupt. 1 Directs the shared message signaled interrupt to processor core 0 by causing the *cint0* output signal from the PIC to assert. See Section 11.1.2, “Interrupts to the Processor Core.”
2	CI1	Critical interrupt 1. Reserved in single-processor implementations. 0 Processor core 1 does not receive this interrupt. 1 Directs the shared message signaled interrupt to processor core 1 by causing the *cint1* output signal from the PIC to assert. See Section 11.1.2, “Interrupts to the Processor Core.”
3–29	—	Reserved, should be cleared.
30	P1	Processor core 1. Indicates whether processor core 1 receives the interrupt through *int.* 0 Processor core 1 does not receive this interrupt. 1 Directs the interrupt to processor core 1 through the assertion of *int1*. **Note:** Reserved in single-processor implementations.
31	P0	Processor core 0. Indicates whether processor core 0 receives the interrupt. 0 Processor core 0 does not receive this interrupt. 1 Directs the interrupt to processor core 0 through the assertion of *int0*. The default destination is for processor core 0 to receive this shared message signaled interrupt after the PIC is reset.

11.3.7 Interrupt Source Configuration Registers

The interrupt source configuration registers control the source and destinations of interrupts, specifying parameters such as the interrupting event, signal polarity, and relative priority.

Figure 11-37 shows the destination registers.

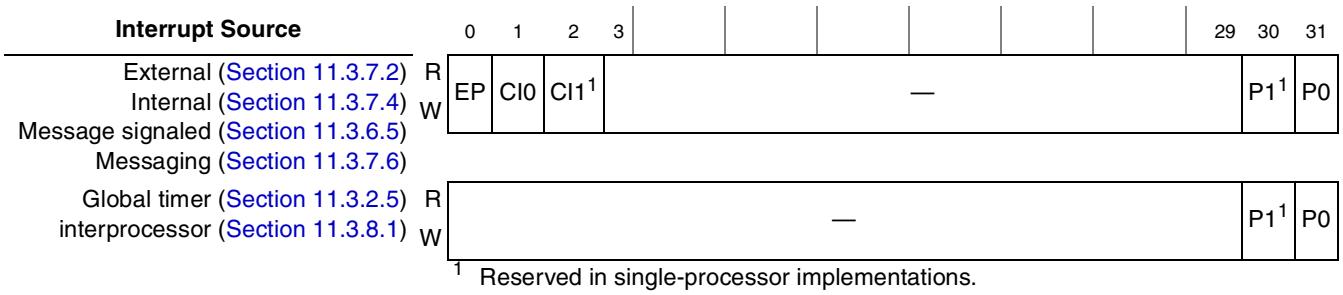


Figure 11-37. Destination Register Summary

Note the following:

- The global timer and interprocessor destination register support only the P0 and P1 options. That is, they cannot be routed to *cint* or to $\overline{\text{IRQ\_OUT}}$.
- Only the global timer and interprocessor interrupts are multicasting, so only these interrupts allow more than one destination bit to be specified.

Figure 11-37 shows the vector/priority registers.

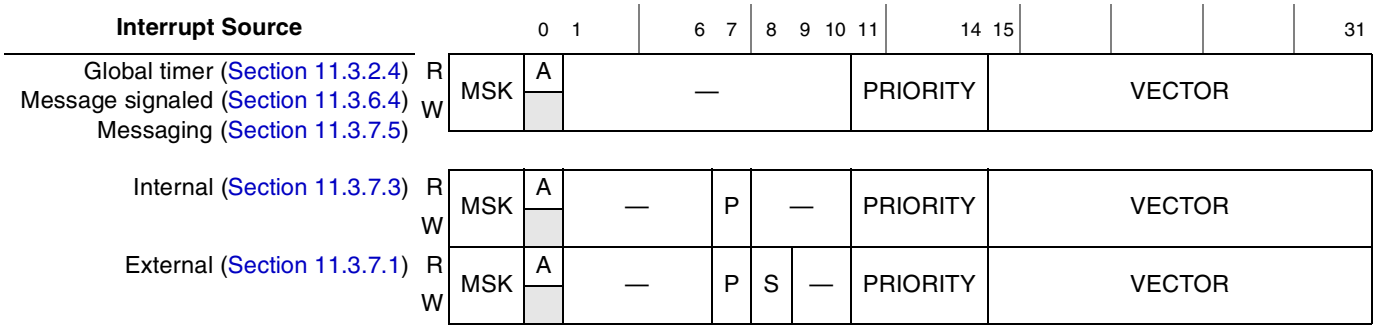


Figure 11-38. Vector/Priority Register Summary

Note the following:

- The MSK, A, PRIORITY, and VECTOR fields have meaning only for interrupts routed to the *int* signal.
- The polarity field, P, is provided to indicate whether the signals from the corresponding source are active high or low.
- The sense field, S, is provided to allow external interrupt sources to be configured as level-sensitive so they can be routed to either *cint* or $\overline{\text{IRQ\_OUT}}$.

11.3.7.1 External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11)

The EIVPRs, shown in Figure 11-39, contain polarity and sense fields for the external interrupts, that is, those caused by the assertion of any of IRQ[0:11]. See Section 11.4.1, "Flow of Interrupt Control," for information on IPR and ISR.

Offset EIVPR0: 0x0000; EIVPR1: 0x0020; EIVPR2: 0x0040; EIVPR3: 0x0060; EIVPR4: 0x5_0080; EIVPR5: 0x5_00A0; EIVPR6: 0x5_00C0; EIVPR7: 0x5_00E0; EIVPR8: 0x5_0100; EIVPR9: 0x5_0120; EIVPR10: 0x5_0140; EIVPR11: 0x5_0160 — Access: Mixed

	0	1	2	7	8	9	10	11	12	15	16	31
R	MSK	A	—		P	S	—		PRIORITY		VECTOR	
W												
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 0 0 0 0	

Figure 11-39. External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11)

Table 11-39 describes the EIVPR fields.

Table 11-39. EIVPR*n* Field Descriptions

Bits	Name	Description
0	MSK	Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*. 0 An interrupt request is generated if the corresponding IPR bit is set. 1 Further interrupts from this source are disabled.
1	A	Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*. 0 No current interrupt activity associated with this source. 1 The interrupt field for this source is set in the IPR or ISR.
2–7	—	Reserved, should be cleared.
8	P	Polarity. Specifies the polarity for the external interrupt. 0 Polarity is active-low or negative edge-triggered. 1 Polarity is active-high or positive edge-triggered.
9	S	Sense. Specifies the sense for external interrupts. 0 The external interrupt is edge sensitive. 1 The external interrupt is level sensitive. This setting must be used to direct the interrupt to $\overline{\text{IRQ\_OUT}}$ or *cint*. Note: If an IRQ*n* signal is used to receive INT*x* signals from one of the PCI Express ports as a root complex, S must be set to be level-sensitive.
10–11	—	Reserved, should be cleared.
12–15	PRIORITY	Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*.
16–31	VECTOR	Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 11-51.

11.3.7.2 External Interrupt Destination Registers (EIDR0–EIDR11)

The EIDRs, shown in Figure 11-40, control the destination of external interrupts caused by the assertion of any of IRQ[0:11]. Only one destination bit may be set; otherwise, behavior is undefined.

Offset EIDR0: 0x0010; IDR1: 0x0030; EIDR2: 0x0050; EIDR3: 0x0070; EIDR4: 0x0090; EIDR5: 0x5_00B0; EIDR6: 0x5_00D0; EIDR7: 0x5_00F0; EIDR8: 0x5_0110; EIDR9: 0x5_0130; EIDR10: 0x5_0150; EIDR11: 0x5_0170 — Access: Mixed

	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
R	EP	CI0	CI1[1]	—																												P0
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 11-40. External Interrupt Destination Registers (EIDRs)

1

Table 11-40 describes the EIDR*n* fields. Because external interrupts can be channeled only to processor 0, the P0 bit is permanently set. As shown in Figure 11-51, if either the CI or EP bits are set, the interrupt is not sent to the processor's interrupt input.

Table 11-40. EIDR*n* Field Descriptions

Bits	Name	Description
0	EP	External signal. Allows interrupt to be serviced externally. EP should be set only for level-sensitive external interrupts (EIVPR*n*[S]= 1). Setting for edge-sensitive does not provide reliable interrupt response. 0 Interrupt is not routed to $\overline{\text{IRQ\_OUT}}$. 1 Interrupt is routed to $\overline{\text{IRQ\_OUT}}$ for external service.
1	CI0	Critical interrupt 0. Ci*n* fields should be set only for level-sensitive external interrupts (EIVPR*n*[S]= 1). Setting them for edge-sensitive does not provide reliable interrupt response. 0 Processor core 0 does not receive this interrupt. 1 Directs the external interrupt to processor core 0 by causing the *cint0* output signal from the PIC to assert. See Section 11.1.2, "Interrupts to the Processor Core."
2	CI1	Critical interrupt 1. Ci*n* fields should be set only for level-sensitive external interrupts (EIVPR*n*[S]= 1). Setting them for edge-sensitive does not provide reliable interrupt response. Reserved in single-processor implementations. 0 Processor core 1 does not receive this interrupt. 1 Directs the external interrupt to processor core 1 by causing the *cint1* output signal from the PIC to assert. See Section 11.1.2, "Interrupts to the Processor Core."
3–30	—	Reserved, should be cleared.
31	P0	Processor core 0. Indicates whether processor core 0 receives the interrupt. 0 Processor core 0 does not receive this interrupt. 1 Directs the interrupt to processor core 0 through the assertion of *int0*. The default destination is for processor core 0 to receive this external interrupt after the PIC is reset.

11.3.7.3 Internal Interrupt Vector/Priority Registers (IIVPR*n*)

The IIVPRs, shown in Figure 11-41, have the same fields and format as the GTVPRs, except that they apply to the internal interrupt sources listed in Table 11-3. These interrupts are all level-sensitive. See Section 11.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

NOTE

Because all internal interrupts are active-high, clearing the polarity field, IIVPR*n*[P], disables that interrupt. Care should be taken to ensure this field is set during initialization and that it is not inadvertently corrupted when loading or reloading IIVPRs with priority, mask, or vector data.

Offset IIVPR0–7 0x0200, 0x0220, 0x0240, 0x0260, 0x0280, 0x02A0, 0x02C0, 0x02E0 Access: Mixed
IIVPR8–15 0x0300, 0x0320, 0x0340, 0x0360, 0x0380, 0x03A0, 0x03C0, 0x03E0
IIVPR16–23 0x0400, 0x0420, 0x0440, 0x0460, 0x0480, 0x04A0, 0x04C0, 0x04E0
IIVPR24–31 0x0500, 0x0520, 0x0540, 0x0560, 0x0580, 0x05A0, 0x05C0, 0x05E0
IIVPR32–39 0x0600, 0x0620, 0x0640, 0x0660, 0x0680, 0x06A0, 0x06C0, 0x06E0
IIVPR40–47 0x0700, 0x0720, 0x0740, 0x0760, 0x0780, 0x07A0, 0x07C0, 0x07E0
IIVPR48–55 0x0800, 0x0820, 0x0840, 0x0860, 0x0880, 0x08A0, 0x08C0, 0x08E0, IIVPR56–63 0x0900, 0x0920, 0x0940, 0x0960, 0x0980, 0x09A0, 0x09C0, 0x09E0

	0	1	2–7	8	9–11	12–15	16–31
R	MSK	A	—	P	—	PRIORITY	VECTOR
W							
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 0 0 0 0

Figure 11-41. Internal Interrupt Vector/Priority Registers (IIVPRs)

Table 11-41 describes the IIVPR fields.

Table 11-41. IIVPR*n* Field Descriptions

Bits	Name	Description
0	MSK	Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*. 0 An interrupt request is generated if the corresponding IPR bit is set. 1 Further interrupts from this source are disabled.
1	A	Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*. 0 No current interrupt activity associated with this source. 1 The interrupt field for this source is set in the IPR or ISR.
2–7	—	Reserved, should be cleared.
8	P	Polarity. Specifies the polarity for the internal interrupt. Note: Because all internal interrupts are active-high, clearing this bit disables the interrupt. 0 Interrupt polarity is active-low. This value disables the interrupt. 1 Interrupt polarity is active-high.
9–11	—	Reserved, should be cleared.
12–15	PRIORITY	Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*.
16–31	VECTOR	Vector (Affects only interrupts routed to *int*). Contains the value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 11-51.

11.3.7.4 Internal Interrupt Destination Registers (IIDR*n*)

The IIDRs, shown in Figure 11-42, have the same fields and format as EIVDRs, except that they apply to the internal interrupt sources listed in Table 11-3. Only one destination bit may be set; otherwise, behavior is undefined.

Offset IIDR0–70x0210, 0x0230, 0x0250, 0x0270, 0x0290, 0x02B0, 0x02D0, 0x02F0 Access:
IIDR8–15 0x0310, 0x0330, 0x0350, 0x0370, 0x0390, 0x03B0, 0x03D0, 0x03F0 Read/Write
IIDR16–23 0x0410, 0x0430, 0x0450, 0x0470, 0x0490, 0x04B0, 0x04D0, 0x04F0
IIDR24–31 0x0510, 0x0530, 0x0550, 0x0570, 0x0590, 0x05B0, 0x05D0, 0x05F0
IIDR32–39 0x0610, 0x0630, 0x0650, 0x0670, 0x0690, 0x06B0, 0x06D0, 0x06F0
IIDR40–47 0x0710, 0x0730, 0x0750, 0x0770, 0x0590, 0x07B0, 0x07D0, 0x07F0
IIDR48–53 0x0810, 0x0830, 0x0850, 0x0870, 0x0890, 0x08B0, 0x08D0, 0x08F0
IIDR54–63 0x0910, 0x0930, 0x0950, 0x0970, 0x0990, 0x09B0, 0x09D0, 0x09F0

	0	1	2																												29	30	31
R W	EP	CI0	CI1[1]	—																												P1[1]	P0
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	1

[1] Reserved in single-processor implementations.

Figure 11-42. Internal Interrupt Destination Registers (IIDRs)

Table 11-42 describes the IIDR fields.

Table 11-42. IIDR*n* Field Descriptions

Bits	Name	Description
0	EP	External signal. Allows internal interrupt to be serviced externally. 0 Interrupt is not routed to $\overline{\text{IRQ\_OUT}}$. 1 Interrupt is routed to $\overline{\text{IRQ\_OUT}}$ for external service.
1	CI0	Critical interrupt 0. See Section 11.1.2, “Interrupts to the Processor Core,” for more information. 0 Processor core 0 does not receive this interrupt. 1 Directs the internal interrupt to processor core 0 by causing the *cint0* output signal from the PIC to assert.
2	CI1	Critical interrupt 1. See Section 11.1.2, “Interrupts to the Processor Core,” for more information. Reserved in single-processor implementations. 0 Processor core 1 does not receive this interrupt. 1 Directs the internal interrupt to processor core 1 by causing the *cint1* output signal from the PIC to assert.
3–29	—	Reserved, should be cleared.
30	P1	Processor core 1. Indicates whether processor core 1 receives the interrupt through *int.* 0 Processor core 1 does not receive this interrupt. 1 Directs the interrupt to processor core 1 through the assertion of *int1*. **Note:** Reserved in single-processor implementations.
31	P0	Processor core 0. Indicates whether processor core 0 receives the interrupt. 0 Processor core 0 does not receive this interrupt. 1 Directs the interrupt to processor core 0 through the assertion of *int0*. The default destination is for processor core 0 to receive this external interrupt after the PIC is reset.

11.3.7.5 Messaging Interrupt Vector/Priority Registers (MIVPR*n*)

The MIVPRs have the same fields and format as the GTVPRs, except they apply to messaging interrupts. See Section 11.4.1, “Flow of Interrupt Control,” for information on IPR and ISR.

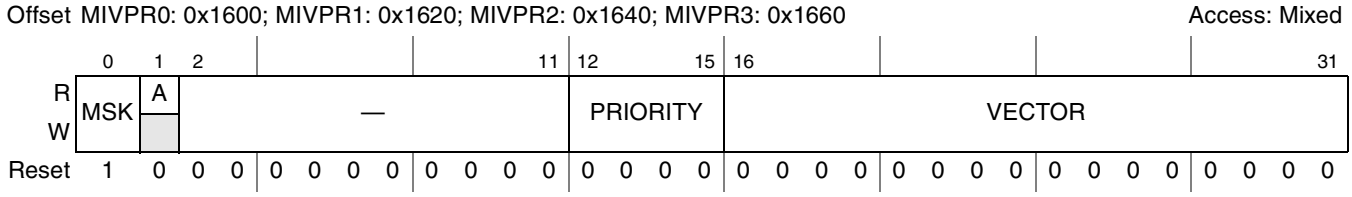


Figure 11-43. Messaging Interrupt Vector/Priority Registers (MIVPR*n*)

Table 11-43 describes the MIVPR*n* fields.

Table 11-43. MIVPR*n* Field Descriptions

Bits	Name	Description
0	MSK	Mask. Mask interrupts from this source. MSK affects only interrupts routed to *int*. 0 An interrupt request is generated if the corresponding IPR bit is set. 1 Further interrupts from this source are disabled.
1	A	Activity. Indicates an interrupt has been requested or is in service. The VECTOR and PRIORITY values should not be changed while this bit is set. Affects only interrupts routed to *int*. 0 No current interrupt activity associated with this source. 1 The interrupt field for this source is set in the IPR or ISR.
2–11	—	Reserved, should be cleared.
12–15	PRIORITY	Priority. Specifies the interrupt priority. The lowest priority is 0 and the highest priority is 15. A priority level of 0 inhibits signalling of this interrupt to the core. Affects only interrupts routed to *int*.
16–31	VECTOR	Vector (Affects only interrupts routed to *int*). Contains value returned when IACK is read and this interrupt resides in the corresponding interrupt request register (IRR) for that core, as shown in Figure 11-51.

11.3.7.6 Messaging Interrupt Destination Registers (MIDR0–MIDR3)

The messaging interrupt destination registers (MIDRs), shown in Figure 11-44, control the destination for the messaging interrupts. Only one destination bit may be set; otherwise, behavior is undefined.

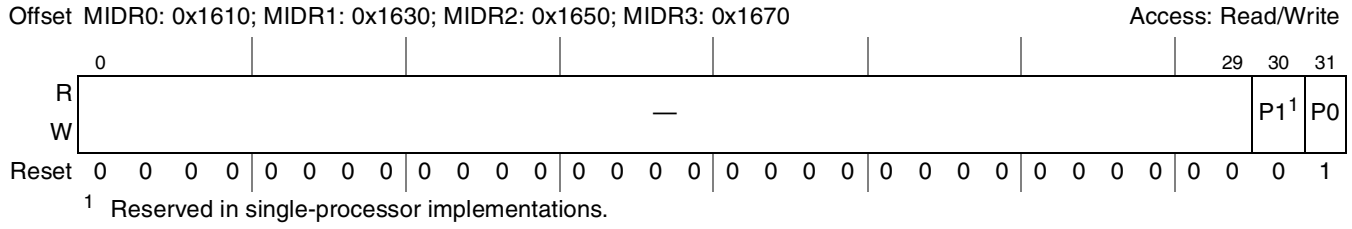


Figure 11-44. Messaging Interrupt Destination Registers (MIDR*n*)

Table 11-44 describes the MIDR*n* fields.

Table 11-44. MIDR*n* Field Descriptions

Bits	Name	Description
0–29	—	Reserved, should be cleared.
30	P1	Processor core 1. Indicates whether processor core 1 receives the interrupt through *int*. 0 Processor core 1 does not receive this interrupt. 1 Directs the interrupt to processor core 1 through the assertion of *int1*. **Note:** Reserved in single-processor implementations.
31	P0	Processor core 0. Indicates whether processor core 0 receives the interrupt. 0 Processor core 0 does not receive this interrupt. 1 Directs the interrupt to processor core 0 through the assertion of *int0*. The default destination is for processor core 0 to receive this external interrupt after the PIC is reset.

11.3.8 Per-CPU (Private Access) Registers

The OpenPIC programming model supports multiprocessor systems of up to 32 separate processors. As such, the OpenPIC interface specification provides for coordinating both the requesting and servicing of interrupts among several processor cores within a single integrated device. To comply with the OpenPIC specification, the PIC incorporates several of these multiprocessor capabilities.

NOTE

Note that these registers are meaningful only for interrupts routed to *int*.

The registers in Table 11-45 are called per-CPU registers because they are duplicated for each core in a multi-core device. The OpenPIC interface specifies that a copy of these registers be available to each core at the same physical address by using the ID of the processor core that initiates the transaction to determine the set of per-CPU registers to access.

Table 11-45. Per-CPU Registers—Private Access Address Offsets

Register Name	Offset
Interprocessor 0 dispatch register (IPIDR0)	0x0040
Interprocessor 1 dispatch register (IPIDR1)	0x0050
Interprocessor 2 dispatch register (IPIDR2)	0x0060
Interprocessor 3 dispatch register (IPIDR3)	0x0070
Current task priority register (CTPR)	0x0080
Who am I register (WHOAMI0)	0x0090
Interrupt acknowledge register (IACK)	0x00A0
End of interrupt register (EOI)	0x00B0

These addresses, shown in Table 11-45, appear in the memory map at the same offset for every processor in what is called the private access space. This duplication allows user code to execute correctly in an multiprocessor environment without needing to know which core it is running on. On a single-core device,

each register has two addresses, one in the normal address space and one in the private access space. It is included on even single-core devices to simplify the porting of such code.

Figure 11-45 shows how the duplicated registers are addressed in a four-core device Note that when accessing a register normally, each core sources a different address. However, when accessing the same register using the per-CPU address space, each core sources the same address.

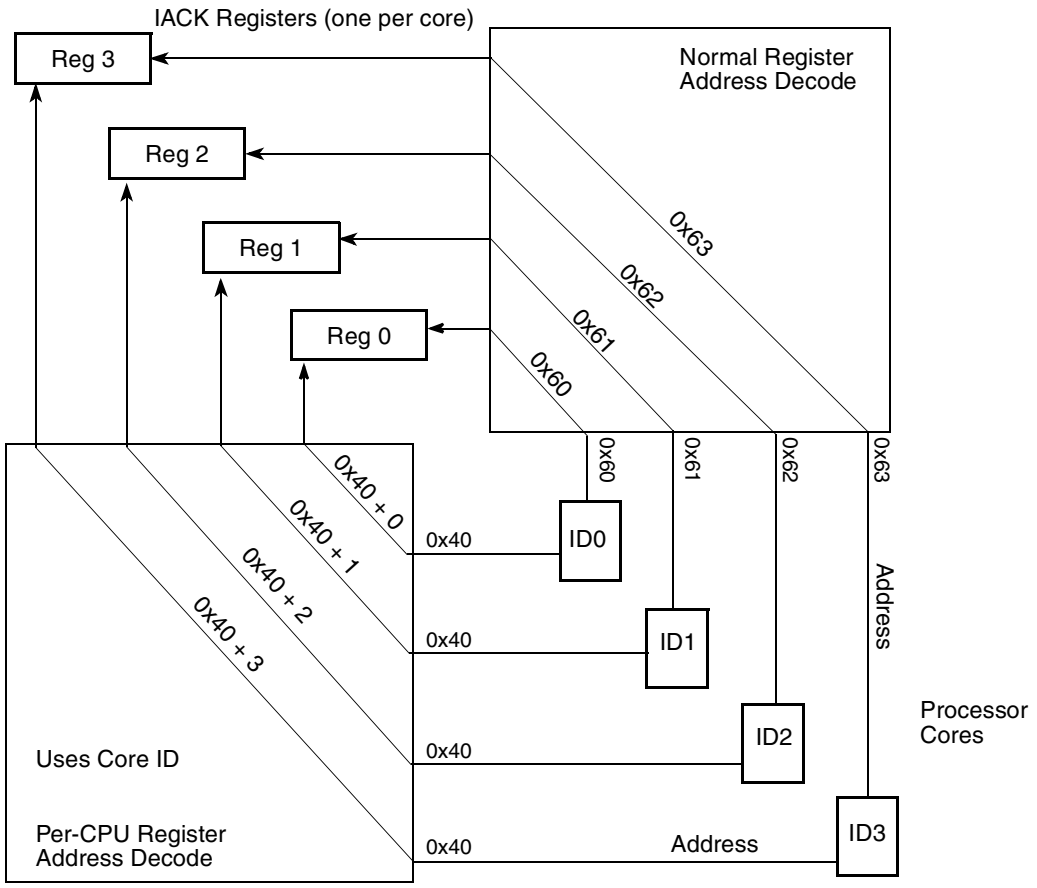
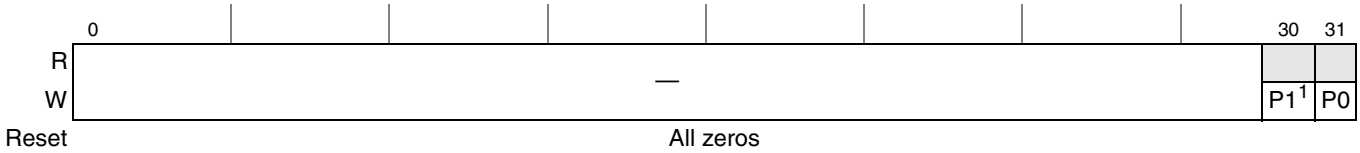


Figure 11-45. Per-CPU Register Address Decoding in a Four-Core Device

11.3.8.1 Interprocessor Interrupt Dispatch Register (IPIDR0–IPIDR3)

Figure 11-46 shows the four IPIDRs, one for each interprocessor interrupt channel. Writing to an IPIDR with a bit set causes a self interrupt for a single-core device. Because external bus masters can write to these registers, this feature can serve as a doorbell type interrupt.

Offset Processor core 0: IPIDR0: 0x0040; IPIDR1: 0x0050; IPIDR2: 0x0060; IPIDR3: 0x0070 Access: Write Only
Processor core 1[1]:IPIDR0: 0x1040; IPIDR1: 0x1050; IPIDR2: 0x1060; IPIDR3: 0x1070
Pre-CPU offsets: IPIDR0: 0x0040; IPIDR1: 0x0050; IPIDR2: 0x0060; IPIDR3: 0x0070



[1] Reserved in single-processor implementations.

Figure 11-46. Interprocessor Interrupt Dispatch Registers (IPIDR0–IPIDR3)

Table 11-41 describes the IPIDR*n* fields.

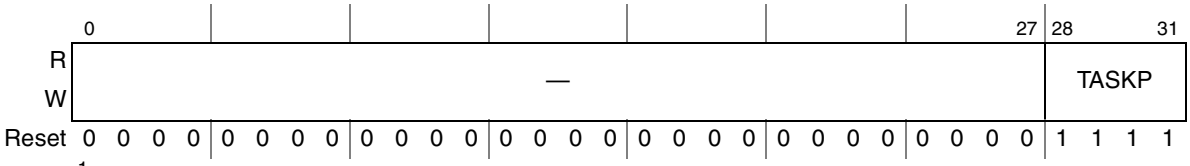
Table 11-46. IPIDR*n* Field Descriptions

Bits	Name	Description
0–29	—	Reserved, should be cleared.
30	P1	Processor core 1. Specifies if processor core 1 receives the interrupt. This interrupt is multicasting, so both P0 and P1 can be set. 0 Processor core 1 does not receive the interrupt 1 Directs the interrupt to processor core 1 **Note:** Reserved in single-processor implementations.
31	P0	Processor core 0. Determines if processor core 0 receives the interrupt. 0 Processor core 0 does not receive the interrupt. 1 Directs the interrupt to processor core 0.

11.3.8.2 Processor Core Current Task Priority Registers 0–1 (CTPR0–CTPR1)

There is one CTPR per processor core on this device as shown in Figure 11-47.

Offset CTPR0: 0x0080; CTPR1[1] 0x1080; Pre-CPU offset: 0x0080 Access: Read/Write



[1] Reserved in single-processor implementations.

Figure 11-47. Processor Core Current Task Priority Registers (CTPR*n*)

NOTE

CTPR has meaning only for interrupts routed to *int*.

Software must write the priority of the current processor core task in the CTPR for each core. The PIC uses this value for comparison with the priority of incoming interrupts. Given several concurrent incoming interrupts, the highest priority interrupt is asserted to that core if the following apply:

- The interrupt is not masked.
- The priority of the interrupt is higher than the values in the corresponding CTPR[TASKP] and ISR.

Table 11-47 describes the CTPR task priority field.

Table 11-47. CTPR*n* Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared.
28–31	TASKP	Task priority. Indicates the threshold that individual interrupt priorities must exceed for the interrupt request to be serviced. 0000–1111 *x*VPR*n*[PRIORITY] must exceed this value for the interrupt request to be serviced. Note the following special cases: 0000 Lowest priority. All interrupts except those whose priority are 0 can be serviced. 1111 Highest priority. No interrupts are signaled to that processor core. Hardware selects this value on a device hard reset or when the corresponding PIR[P*n*] is set.

11.3.8.3 Who Am I Registers 0–1 (WHOAMI0–WHOAMI1)

The processor core WHOAMI*n* register, shown in Figure 11-48, can be read by a processor core to determine its physical connection to the PIC. The value returned when reading this register may be used to determine the value for the destination masks used for dispatching interrupts.

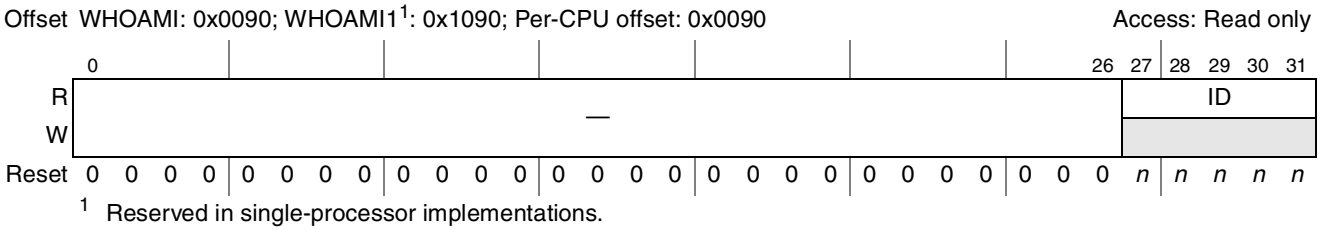


Figure 11-48. Processor Core Who Am I Registers (WHOAMI*n*)

Table 11-48 describes the WHOAMI*n* fields.

Table 11-48. WHOAMI*n* Field Descriptions

Bits	Name	Description
0–26	—	Reserved, should be cleared.
27–31	ID	Returns the ID of the processor core reading this register. 0_0000 Processor core 0 0_0001 Processor core 1. (Value not supported in single-processor implementations.) 1_1111 Other devices

11.3.8.4 Processor Core Interrupt Acknowledge Registers 0–1 (IACK0–IACK1)

NOTE

IACK has meaning only for interrupts routed to *int* and should not be accessed for interrupts routed to *cint* or $\overline{\text{IRQ\_OUT}}$.

In systems based on processors built on Power Architecture™ technology, the interrupt acknowledge function occurs as an explicit read operation to a memory-mapped interrupt acknowledge register (IACK), shown in Figure 11-49. Each processor core has an IACK register assigned to it. Reading IACK returns the interrupt vector corresponding to the highest priority pending interrupt. Reading IACK also has the following side effects:

- The associated field in the corresponding interrupt pending register (IPR) is cleared for edge-sensitive interrupts. See Section 11.4.1.2, “Interrupts Routed to int.”
- The corresponding in-service register (ISR) is updated.
- The corresponding *int* output signal from the PIC is negated.

Reading IACK when no interrupt is pending returns the spurious vector value, as described in Section 11.3.1.9, “Spurious Vector Register (SVR).”

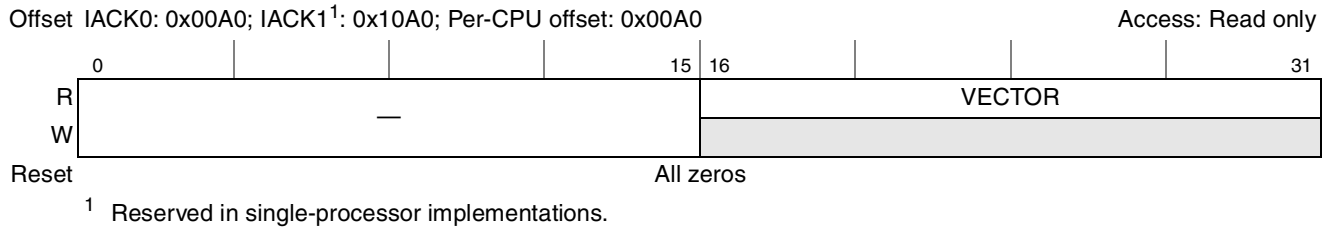


Figure 11-49. Processor Core Interrupt Acknowledge Registers (IACK*n*)

Table 11-49 describes the IACK*n* fields.

Table 11-49. IACK*n* Field Descriptions

Bits	Name	Description
0–15	—	Reserved, should be cleared.
16–31	VECTOR	Interrupt vector. Vector of the highest pending interrupt (read only)

11.3.8.5 Processor Core End of Interrupt Registers (EOI0–EOI1)

NOTE

EOI has meaning only for interrupts routed to *int* and should not be accessed for interrupts routed to *cint* or $\overline{\text{IRQ\_OUT}}$.

Each core is assigned an EOI register, shown in Figure 11-50. Writing to EOI signals the end of processing for the highest-priority interrupt (routed to *int*) currently in service. It also updates the corresponding ISR*n* by retiring the highest priority interrupt. Data values written to EOI are ignored, and zero is assumed.

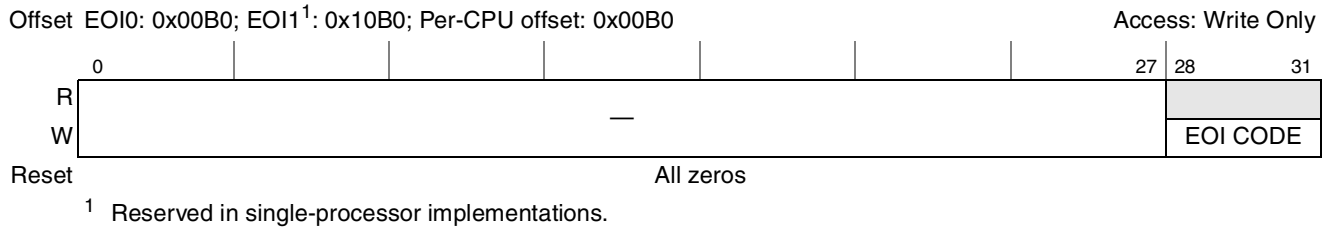


Figure 11-50. End of Interrupt Registers (EOI*n*)

Table 11-50 describes the EOI*n* fields.

Table 11-50. EOI*n* Field Descriptions

Bits	Name	Description
0–27	—	Reserved, should be cleared.
28–31	EOI CODE	0000 (write only)

11.4 Functional Description

This section is a functional description of the PIC.

11.4.1 Flow of Interrupt Control

Figure 11-51 shows the flow of interrupts directed by the PIC to the *int*, *cint*, and $\overline{\text{IRQ\_OUT}}$ outputs. Note that this diagram describes a conceptual model of an PIC on a single processor. This logic is replicated for each implemented processor. This conceptual diagram does not fully represent all internal circuitry of the implementation.

This figure focusses especially on the OpenPIC-defined logic and shows how the PIC controls interrupt requests that target the *int* signal. The flow in Figure 11-51 is from the bottom to the top, and shows at the bottom how the destination register associated with each source determines the path.

11.4.1.1 Interrupts Routed to *cint* or $\overline{\text{IRQ\_OUT}}$

Interrupt requests routed to *cint* or $\overline{\text{IRQ\_OUT}}$ bypass the logic that is dedicated to interrupt sources that compete for *int*. That is, if *x*IDR*n*[CI*n*] or *x*IDR*n*[EP] = 1, corresponding *x*IVPR field settings have no hardware effects; however, an interrupt handler may be able to make use of some of those fields.

cint signals are connected to the respective core's machine check input.

NOTES

Because interrupt sources routed to *cint* or $\overline{\text{IRQ\_OUT}}$ must be level sensitive, EIVPR[S] should be set. See Section 11.3.7.1, "External Interrupt Vector/Priority Registers (EIVPR0–EIVPR11)."

Because these interrupts bypass the OpenPIC logic, it is especially important that handlers do not read IACK. Doing so causes a spurious interrupt. Likewise, they should not write EOI.

11.4.1.2 Interrupts Routed to *int*

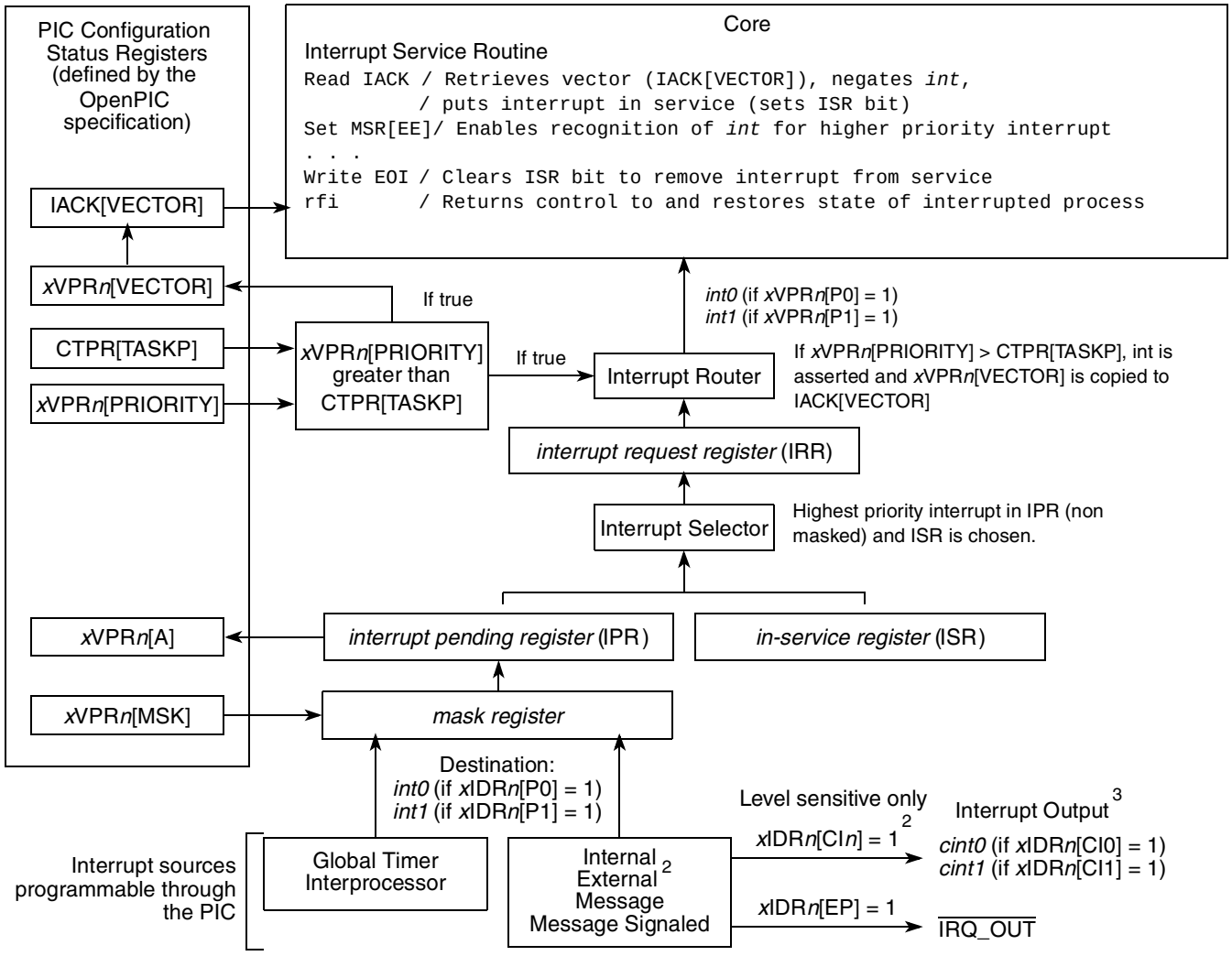
As shown in Figure 11-51, the PIC receives interrupt requests from external and internal sources and from within the PIC itself. As Figure 11-51 shows, all of these interrupt sources can be routed to *int*; the global timer and timer processor interrupts can be directed only to *int*.

The sources' mask bits (*x*VPR*n*[MSK]) are tracked in the internal mask register. If a source's MSK bit is set, the mask register prevents the PIC from asserting *int* on its behalf.

Unmasked interrupt requests are qualified and latched in the interrupt pending register (IPR), an internal interrupt summary register with a bit for each source. If an interrupt request is multi-cast, a bit is set in the IPR for each targeted processor. Although the IPR cannot be read by software, when an IPR bit is set, the corresponding source's activity bit (*x*VPR*n*[A]) is automatically set.

The interrupt selector monitors the IPR and the in-service register (ISR), which tracks previously taken interrupts that were superseded by a higher-priority interrupt before the interrupt handler finished. The interrupt selector recognizes the highest priority unmasked interrupt request and latches it into the interrupt request register (IRR). The source's vector (*x*VPR*n*[VECTOR]) is copied to IACK[VECTOR], which the interrupt handler retrieves by reading IACK.

If the priority (*x*VPR*n*[PRIORITY]) of an interrupt latched in the IRR is higher than the value in the target processor's CTPR[TASKP], the interrupt router asserts the external interrupt signal (*int*), causing that processor core to vector to its external interrupt handler.



1 If *cint* or $\overline{\text{IRQ\_OUT}}$ is the destination, EIVPR*n*[S] must be set to configure the source as level sensitive.

2 If multiple destination register bits are set, PIC behavior is undefined.

3 Although setting CI*n* directs the interrupt request to the critical interrupt output (*cint0*/*cint1*), integrated logic may connect this signal to a different interrupt input to the core.

Figure 11-51. PIC Interrupt Processing Flow Diagram for Each Core (*n*)

The interrupt handler must acknowledge the interrupt by explicitly reading the corresponding IACK register, described in Section 11.3.8.4, "Processor Core Interrupt Acknowledge Registers 0–1 (IACK0–IACK1)." The PIC interprets this read as an interrupt acknowledge (IACK) cycle. The IACK cycle not only returns the source's vector, it also negates the *int* signal to the processor (making it possible for a higher priority interrupt to assert *int*) and sets the source's bit in the ISR, indicating that this interrupt has been put in service. An interrupt remains in service from the time until the corresponding end-of-interrupt register (EOI) is written, generating what the PIC considers an EOI signal.

Figure 11-51 shows required elements in the interrupt handler

11.4.1.2.1 Interrupt Source Priority

Each interrupt source routed to *int* is assigned a value through its *x*VPR*n*[PRIORITY] field. Priority values range from 0 to 15, where 15 is the highest. Interrupts are delivered only when the priority of the source is greater than the destination processor's CTPR[TASKP]. Therefore, setting *x*VPR*n*[PRIORITY] to zero inhibits that interrupt. Likewise, setting TASKP to 15 prevents the PIC from delivering interrupts to that core through the *int* signal. Note that this is the reset value, preventing the PIC from asserting *int* before the PIC is configured.

The PIC services simultaneous interrupts occurring with the same priority according to the following order:

1. MSG0–MSG3
2. MSI0–MSI7
3. IPI0–IPI3
4. Group A timer0–timer3
5. Group B timer0–timer3
6. IRQ[0:11]/PCI INT*x*
7. Internal0–internal63

For example, if MSG0, MSG2, and IPI0 are all assigned the same priority and receive simultaneous interrupts, they are serviced in the following order:

1. MSG0
2. MSG2
3. IPI0

11.4.1.2.2 Interrupt Acknowledge

When the PIC causes *int* to be asserted, the external interrupt service routine acknowledges the request by reading that core's IACK register, which at this point holds the 16-bit vector value for the interrupt source that generated the request. This is the value programmed in that source's *x*VPR*n*[VECTOR]. Reading IACK has the following effects:

- The *int* signal for that core is negated, making it possible for another interrupt source to signal an external interrupt to the core, and more particularly, allowing the PIC to signal a higher-priority interrupt, as described in Section 11.4.1.2.4, "Nesting of Interrupts."
- The source that caused that resource is represented in the internal in-service register (ISR).

The interrupt is then considered to be in service. It remains so until the processor core performs a write to the corresponding EOI. Writing to EOI is referred to as an EOI cycle.

11.4.1.2.3 Spurious Vector Generation

Under certain circumstances, the PIC has no valid vector to return to a processor core during an interrupt acknowledge cycle. In these cases, the spurious vector from the spurious vector register is returned. The following cases cause a spurious vector fetch:

- *int* is asserted in response to an externally or internally-sourced interrupt which is activated with level-sensitive logic, and the asserted level is negated before the interrupt is acknowledged.
- *int* is asserted for an interrupt source that is later masked (using the mask bit in the vector/priority register corresponding to that source) before the interrupt is acknowledged.
- *int* is asserted for an interrupt source that is later masked by an increase in the task priority level before the interrupt is acknowledged.
- An interrupt acknowledge cycle is performed by the processor core in spite of the fact that the int signal has not been asserted by the PIC.

In all cases, a spurious vector is returned only if no pending interrupt has sufficient priority to signal an interrupt, otherwise, the vector for that interrupt source is returned.

NOTE

EOI should not be written in response to a spurious vector. Otherwise, a previously accepted interrupt might be cleared unintentionally.

11.4.1.2.4 Nesting of Interrupts

While an interrupt is being handled, if an interrupt request arrives with a higher *x*VPR*n*[PRIORITY] value, the interrupt being serviced is superseded. As described in Section 11.4.1.2, "Interrupts Routed to *int*," the PIC asserts *int*, and the newer, higher priority interrupt is handled. This happens even if software, as part of its interrupt service routine, updates the corresponding CTPR with a lower value.

Thus, although several interrupts can be in service simultaneously (and tracked by the ISR), the highest priority interrupt by that processor is always the one actively handled. When the interrupt routine completes, it performs a write EOI cycle, a side effect of which is to take the current highest priority interrupt out of service (removes it from the ISR). At this point, the interrupt selector chooses the new highest priority interrupt request, and, assuming CTPR[TASKP] has not been updated to a value higher than the new interrupt, the PIC asserts *int* on its behalf.

The next write EOI cycle takes the current highest priority interrupt out of service. An interrupt with lower priority than those in service is not started until all higher priority interrupts complete even if its priority is greater than the CTPR value.

11.4.2 Interprocessor Interrupts

Processors 0 and 1 can generate interprocessor interrupts that target either or both processors. A self interrupt occurs when a core dispatches an interprocessor interrupt event to itself. Interrupts are initiated by writing either or both of the PO*n* bits in an interprocessor interrupt dispatch register (IPIDR0–IPIDR3) of one of the four IPI channels. If subsequent interprocessor interrupts from a given channel to a given target processor are initiated before the first is acknowledged, only one interrupt is generated.

11.4.3 Message Interrupts

The four MSGRs, described in Section 11.3.5.1, "Message Registers (MSGR0–MSGR3)," can be used to send 32-bit messages to one or more processors. A messaging interrupt is generated by writing an MSGR if the corresponding MER bit is set and the interrupt is not masked. Reading a MSGR or writing a 1 to the status bit clears the interrupt.

11.4.4 Shared Message Signaled Interrupts

There are eight shared MSIRs, described in Section 11.3.6.1, "Shared Message Signaled Interrupt Registers (MSIR0–MSIR7)," that indicate which of the interrupt sources sharing the MSI register have pending interrupts. Up to 32 sources can share any individual MSI register. A shared message signaled interrupt is generated by writing to Shared Message Signaled Interrupt Index Register (MSIIR) fields SRS and IBS. This register is primarily intended to support inbound PCI Express message signaled interrupts (MSIs) when the PCI Express controller is configured as a root complex (RC).

MSIIR[SRS] selects the associated MSIR and MSIIR[IBS] selects the interrupt flag/bit in that register that is to be set. The corresponding interrupt needs to be unmasked for the interrupt to occur. A read to an MSIR clears the all of its flags.

11.4.5 PCI Express INTx/IRQ*n* Sharing

Whenever the PCI Express controller is in root complex mode and it receives an inbound INTx asserted or negated message transaction, it asserts or negates an equivalent internal INTx signal to the PIC. This INTx virtual-wire interrupt signaling mechanism replaces the PCI standard sideband interrupts (INTA, INTB, INTC, and INTD) that historically were connected to the IRQ*n* external interrupt inputs. The internal INTx signals from the PCI Express controller are logically combined with the interrupt request (IRQ*n*) signals so that they share the same OpenPIC external interrupt controlled by the associated EIVPR*n* and EIDR*n* registers.

Table 11-51 details the association of INTx signals to IRQ*n* signals.

Table 11-51. PCI Express INTx/IRQ*n* Sharing

PCI Express Number	INTx	IRQ*n*
PCI Express 1	INTA	IRQ0
	INTB	IRQ1
	INTC	IRQ2
	INTD	IRQ3
PCI Express 2	INTA	IRQ4
	INTB	IRQ5
	INTC	IRQ6
	INTD	IRQ7

In general, these signals should be considered mutually exclusive. If a PCI Express INTx signal is being used, the PIC must be configured so that external interrupts are level sensitive (EIVPRn[S] = 1). If an IRQ*n* signal is being used as edge-triggered (EIVPRn[S] = 0), the system must not allow inbound PCI Express INTx transactions.

Note that it is possible to share IRQ*n* and INTx if the external interrupt is level sensitive; however, if an interrupt occurs, the interrupt service routine must poll both the external sources connected to the IRQ*n* input and the PCI Express INTx sources to determine from which path the external interrupt came. In any case, IRQ*n* should be pulled to the negated state as determined by the associated polarity setting in EIVPR*n*[P].

11.4.6 Global Timers

There are appropriate clock prescalers and synchronizers to provide a time base for the internal PIC timers. These 8 timers are organized as 2 groups of 4 timers each. The timers can be individually programmed to generate a processor core interrupt when they count down to zero and can be used to generate regular periodic interrupts. Each timer has the following four configuration and control registers:

- Global timer current count register (GTCCR*xn*)
- Global timer base count register (GTBCR*xn*)
- Global timer vector-priority register (GTVPR*xn*)
- Global timer destination register (GTDR*xn*)

The timer frequency should be written to the TFRR*xn*, described in Section 11.3.2.1, “Timer Frequency Reporting Register (TFRRA–TFRRB).”

Timer interrupts are all edge-triggered interrupts. If a timer period expires while a previous interrupt from the same source is pending or in service, the subsequent interrupt is lost.

The timer control register (TCR) provides users with the ability to create timers larger than the 31-bit global timers. The timer frequency can also be changed by setting the appropriate TCR fields, as described in Section 11.3.2.6, “Timer Control Registers (TCRA–TCRB).”

11.4.7 Resets

This section describes the behavior of the PIC at reset and the PIC’s ability to initiate processor resets.

11.4.8 Resetting the PIC

The PIC is reset by a device power-on reset (POR) or by software that sets GCR[RST], either of which causes the following:

- All pending and in-service interrupts are cleared.
- All interrupt mask bits are set.
- Polarity, sense, external signal, critical interrupt, and activity fields are reset to default values.
- PIR, TFRR, TCR, MER, MSR, and MSGR0–MSGR3 are cleared.
- MSG and timer destination fields are set.

- The interprocessor dispatch registers are cleared.
- All timer base count values are reset to zero with count inhibited.
- CTPR[TASKP] is reset to 0x000F, disabling delivery of interrupts that target *int*.
- The spurious interrupt vector resets to 0xFFFF.
- The PMMRs are reset to 0xFFFF.
- The PIC defaults to the pass-through mode (GCR[M] = 0).
- All other registers remain at their pre-reset programmed values.

GCR[RST] is automatically cleared when the reset sequence is complete.

11.4.8.1 Processor Core Resetting

A processor core can be reset by writing to the processor core reset register (PRR). This causes the assertion of the *core0_hreset* and/or *core1_hreset* output signals from the PIC. When this occurs, the corresponding CTPR also gets written to 0x000F to disable the delivery of any interrupts to that core.

11.4.8.2 Processor Core Initialization

A software reset can be routed to either of the cores by writing to the processor core initialization register (PIR). This causes the assertion of the corresponding *sreset* output signal from the PIC. When this occurs, the corresponding CTPR also gets written to 0x000F to prevent delivery of any interrupts to *int*.

11.5 Initialization/Application Information

This section contains initialization and application information for the PIC.

11.5.1 Programming Guidelines

The following subsections contain information about programming PIC registers.

11.5.1.1 PIC Registers

Most PIC control and status registers are readable and return the last value written. The exceptions to this rule are as follows:

- Interprocessor dispatch and EOI registers, which return zeros on reads.
- Activity bits (A) of the vector/priority registers reflect the status of the corresponding interrupt source.
- IACK, which returns the vector of the highest priority pending interrupt or the spurious vector (SVR[VECTOR]) if none is pending.
- Reserved fields always return 0.

When the PIC is in mixed mode (GCR[M] = 1), the following guidelines are recommended:

- All PIC registers must be located in a cache-inhibited, guarded area (configured through the core's MMU).

- The PIC portion of the address map must be set up appropriately.

In addition, the following initialization sequence is recommended:

1. Write the vector, priority, and polarity values in each interrupt's vector/priority register, leaving their MSK (mask) bit set. This is required only if interrupts are used.
2. Clear CTPR (CTPR = all zeros).
3. Program the PIC to mixed mode by setting GCR[M].
4. Clear the MSK bit in the vector/priority registers to be used.
5. Perform a software loop to clear all pending interrupts:
   — Load counter with FRR[NIRQ].
   — While counter > 0, read IACK and write EOI to guarantee all the IPR and ISR bits are cleared.
6. Set the processor core CTPR values to the desired values.
7. Set MER to 0x0000_000F. See Section 11.3.5.2, "Message Enable Register (MER)," for more information.

Depending on the interrupt system configuration, the PIC may generate spurious interrupts to clear interrupts latched during power-up. A spurious or non-spurious vector is returned for an interrupt acknowledge cycle in this case. See the programming note below for the non-spurious case.

NOTE

Because the default polarity/sense for external interrupts is edge-sensitive, and edge-sensitive interrupts are not cleared until they are acknowledged, it is possible for the PIC to store spurious edges detected during power-up as pending external interrupts. If software permanently configures an external interrupt source to be edge-sensitive, it may receive the vector for the interrupt source and not a spurious interrupt vector when software clears the mask bit. This can occur once for any edge-sensitive interrupt when its mask bit is first cleared and the PIC is in mixed mode.

To avoid a false interrupt for this case, software can clear the IPR of these spurious edge detections by first configuring the polarity/sense of external interrupt sources to be level-sensitive: high-level if the input is a positive-edge source and low-level if it is a negative-edge source (while the mask bit remains set). After this is complete, configuring the external interrupt source as edge-sensitive does not cause a false interrupt.

11.5.1.2 Changing Interrupt Source Configuration

To change the vector, priority, polarity, sense, or destination of an active (unmasked) interrupt source, the following steps should be taken:

1. Mask the source using the mask (MSK) bit in the vector/priority register.
2. Wait for the activity (A) bit for that source to be cleared.
3. Make the desired changes.
4. Unmask the source.

Note that changing the destination from *int* to *cint* or $\overline{\text{IRQ\_OUT}}$ makes the A, MSK, and PRIORITY fields meaningless.

Chapter 12
I²C Modules

12.1 Overview

This chapter describes the dual inter-integrated circuit (I^2C) bus modules implemented on this device and covers the following topics:

- Section 12.2, “Introduction to I^2C”
- Section 12.3, “Signal Descriptions”
- Section 12.4, “Memory Map/Register Definition”
- Section 12.5, “Functional Description”
- Section 12.6, “Initialization/Application Information”

12.2 Introduction to I^2C

This section presents the following topics:

- Section 12.2.1, “Definition: I^2C Module”
- Section 12.2.2, “Advantages of the I^2C Bus”
- Section 12.2.3, “Module Block Diagram”
- Section 12.2.4, “Features”
- Section 12.2.5, “Modes of Operation”
- Section 12.2.6, “Definition: I^2C Conditions”

12.2.1 Definition: I^2C Module

The I^2C module is a functional unit that provides 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.

12.2.2 Advantages of the I^2C Bus

The two-wire I^2C bus minimizes interconnections between devices. The synchronous, multiple-master I^2C 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.

12.2.3 Module Block Diagram

Figure 12-1 shows a block diagram of the I²C module.

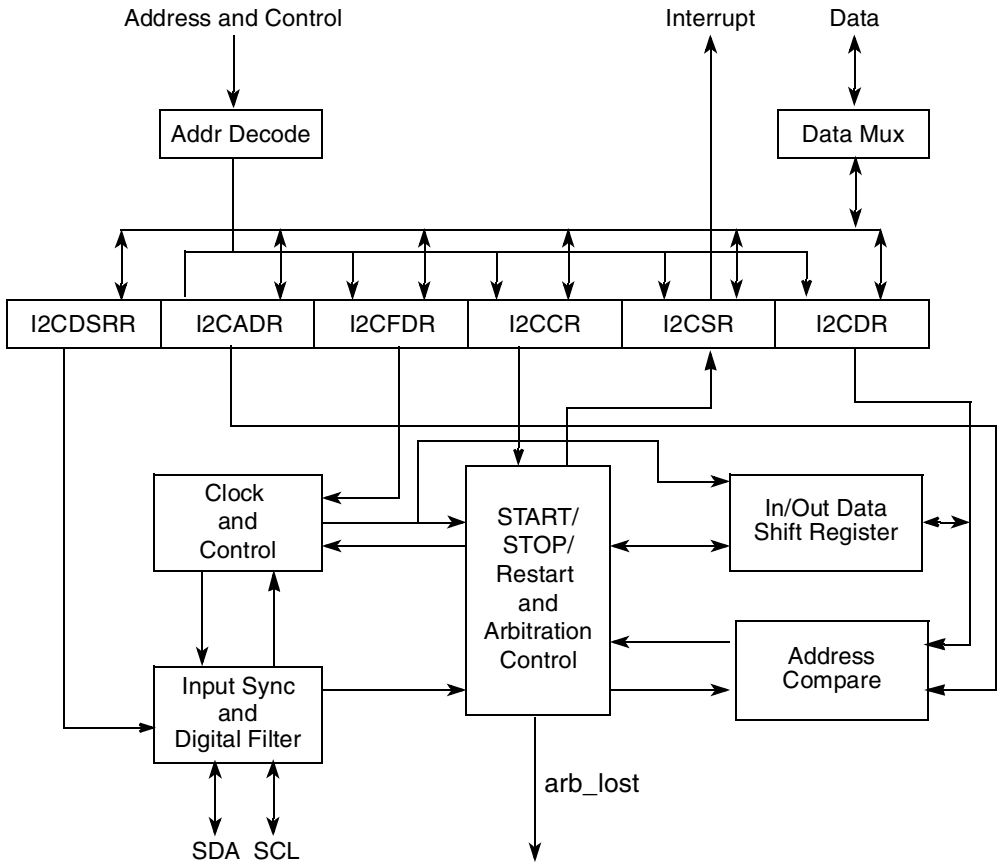


Figure 12-1. I²C Module Block Diagram

12.2.4 Features

The I²C module includes the following features:

- Acknowledge bit generation and detection
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Bus busy detection
- Calling-address identification interrupt
- Multiple-master operation
- On-chip filtering for spikes on the bus
- Software-programmable clock frequency
- Software-selectable acknowledge bit
- START and STOP signal generation and detection
- Two-wire interface

12.2.5 Modes of Operation

The I²C modules can operate in one of several modes, as shown in Table 12-1.

Table 12-1. I²C Module Modes of Operation

Mode	Description	Important Notes
Master mode	The I²C module is the driver of the SDA line.	• Do not use the I²C module's slave address as a calling address. • The I²C module cannot be a master and a slave simultaneously.
Slave mode	The I²C module is not the driver of the SDA line.	• Enable the I²C module before a START condition from a non-I²C master is detected. • By default the I²C module performs as a slave receiver.
Interrupt-driven byte-to-byte data transfer mode	When successful slave addressing is achieved (and the I²C module deasserts SCL), the data transfer can proceed on a byte-to-byte basis in the direction specified by the R/W bit sent by the calling master.	Follow each byte of data by an acknowledge bit, which is signaled from the receiving device.
Boot sequencer mode	This mode can be used to initialize the configuration registers in the device after the I²C1 module is initialized.	The device powers up with boot sequencer mode disabled as a default, but this mode can be selected with the cfg_boot_seq[0:1] power-on reset (POR) configuration signals.

12.2.6 Definition: I²C Conditions

Table 12-2 shows the I²C-specific conditions defined for the I²C module.

Table 12-2. I²C Conditions

Condition	Description
START	A condition that denotes the beginning of a new data transfer and awakens all slaves. Each data transfer contains several bytes of data.
Repeated START	A START condition that is generated without a STOP condition to terminate the previous transfer.
STOP	A condition generated by the master to terminate a transfer and free the bus.

12.3 Signal Descriptions

This section presents the following topics:

- Section 12.3.1, "Signal Overview"
- Section 12.3.2, "Detailed Signal Descriptions"

12.3.1 Signal Overview

The I²C module uses the Serial Data (SDA) and Serial Clock (SCL) signals as a communication interconnect with other devices. The signal patterns driven on the SDA signal represent address, data, or read/write information at different stages of the protocol.

All devices connected to the SDA and SCL signals must have open-drain or open-collector outputs. The logical AND function is performed on both signals with external pull-up resistors. For the electrical characteristics of these signals, see the hardware specifications document for this device.

12.3.2 Detailed Signal Descriptions

The SDA and SCL signals are described in Table 12-3.

Table 12-3. I²C Module—Detailed Signal Descriptions

Signal	I/O	Description	
IIC*n*_SCL	I/O	Serial Clock. Performs as an input when the device is programmed as an I²C slave. SCL also performs as an output when the device is programmed as an I²C master. **Idle State**: High	
	O	When the I²C module is a master, it drives SCL along with SDA when transmitting. As a slave, the I²C module drives SCL low for data pacing. As output for the bidirectional serial clock, SCL operates as described below.	
		State Meaning	Asserted/Negated — SCL is driven along with SDA as the clock for the data.
	I	When the I²C module is idle or is acting as a slave, SCL is an input by default. The I²C module uses SCL to synchronize incoming data on SDA.The bus is assumed to be busy when SCL is detected low. As input for the bidirectional serial clock, SCL operates as described below.	
		State Meaning	Asserted/Negated — The I²C module uses the SCL signal to synchronize incoming data on SDA. The bus is assumed to be busy when SCL is detected low.
IIC*n*_SDA	I/O	Serial Data. Performs as an input when the device is in a receiving mode. SDA also performs as an output signal when the device is transmitting (either as an I²C master or slave). **Idle State**: High	
	O	When the I²C module is writing as a master or slave, it drives data on SDA synchronous to SCL. As output for the bidirectional serial data, SDA operates as described below.	
		State Meaning	Asserted/Negated — Data is driven.
	I	When the I²C module is idle or is in a receiving mode, SDA is an input by default. The unit receives data from other I²C devices on SDA. The bus is assumed to be busy when SDA is detected low. As input for the bidirectional serial data, SDA operates as described below.	
		State Meaning	Asserted/Negated — SDA is used to receive data from other devices. The bus is assumed to be busy when SDA is detected low.

12.4 Memory Map/Register Definition

The I²C module registers and their offsets are described in this section, as well as register access and register figure conventions. This section also contains important notes about the location, byte order, and read/write accesses of registers.

NOTES

- Register Location: Locate all I²C registers in a cache-inhibited page.
- Byte Order: The I²C registers are shown in big-endian format. For a system that is in little-endian mode, the software must swap the bytes

appropriately. Byte swapping is necessary because the I²C registers are byte registers.

- Read/Write Accesses: To guarantee in-order execution, execute a synchronizing assembly instruction after each I²C register read/write access.
- Writes to Reserved Fields: Reserved bits must be written with the value that they returned when read. Program the register by reading its value, modifying the appropriate fields, and writing back the value.

This section is organized as follows:

12.4.1 I²C Memory Map

Table 12-4 lists the I²C registers in offset order and provides links to the complete register descriptions.

Table 12-4. Dual I²C Module Memory Map

I²C Controller 1—Block Base Address 0x0_3000 I²C Controller 2—Block Base Address 0x0_3100				
Offset[1]	**I²C Register**	**Access**	**Reset**	**Section/Page**
I²C1 Registers				
0x000	I2CADR—I²C address register	R/W	0x00	12.4.2/12-6
0x004	I2CFDR—I²C frequency divider register	R/W	0x00	12.4.3/12-6
0x008	I2CCR—I²C control register	Mixed	0x00	12.4.4/12-7
0x00C	I2CSR—I²C status register	Mixed	0x81	12.4.5/12-9
0x010	I2CDR—I²C data register	R/W	0x00	12.4.6/12-10
0x014	I2CDFSRR—I²C digital filter sampling rate register	R/W	0x10	12.4.7/12-11
I²C2 Registers				
0x100–0x114	I²C2 registers **Note:** I²C2 has the same memory-mapped registers described for I²C1 from 0x000 to 0x014, except the I²C2 offsets range from 0x100 to 0x114.			

Note:

1 To calculate a register's absolute address, add CCSRBAR, the I²C module base address, and the register's offset in this column.

12.4.2 I²C Address Register (I2CADR)

The I²C address register (I2CADR), shown in Figure 12-2, specifies the address to which the I²C module responds if the I²C is addressed as a slave. This is not the address sent on the bus during the address-calling cycle when the I²C module is in master mode.

Figure 12-2. I²C Address Register (I2CADR)

The fields in the I2CADR are described in Table 12-5.

Table 12-5. I2CADR Field Descriptions

Bits	Name	Description
0–6	ADDR	Slave address. The ADDR field specifies the slave address to which the I²C module responds if it is addressed as a slave. If the I²C is in slave mode and a transaction's calling address matches I2CADR[ADDR], the module will set I2CSR[MIF], signaling a pending interrupt. (For more information about I2CSR[MIF], see Table 12-8.)
7	—	Reserved

12.4.3 I²C Frequency Divider Register (I2CFDR)

The I²C frequency divider register (I2CFDR), shown in Figure 12-3, specifies the ratio used to prescale the clock for selecting a specific bit rate.

For additional guidance about the proper use of I2CFDR and I2CDFSRR on Power Architecture™ integrated host/communications processors, refer to the application note AN2919, *Determining the I²C Frequency Divider Ratio for SCL*.

NOTE

Writing to the Reserved Field: If you write to the reserved field, always write back the field's original value, as described in the note on page 12-5.

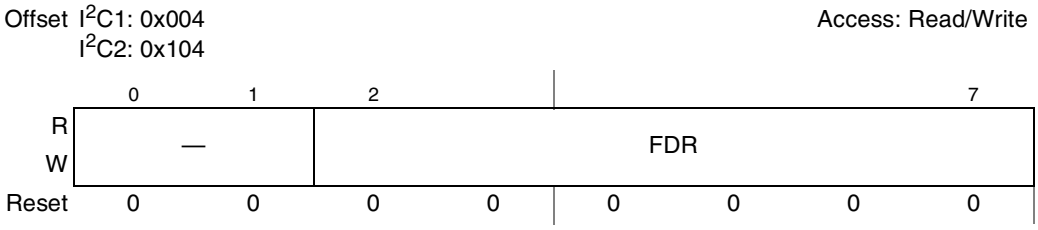


Figure 12-3. I²C Frequency Divider Register (I2CFDR)

The I2CFDR fields and FDR field settings for clock divider values are listed in Table 12-6.

Table 12-6. I2CFDR Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–7	FDR	Frequency divider ratio. Specifies the ratio used to prescale the clock for bit rate selection. The serial bit clock frequency of the SCL is equal to the platform(MPX) clock divided by the designated divider. You can change the frequency divider value at any point in a program. The serial bit clock frequency divider selections are described in the following list:

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				

12.4.4 I²C Control Register (I2CCR)

The I²C control register (I2CCR), shown in Figure 12-4, contains fields for controlling several actions, modes, messages, conditions, and capabilities.

NOTE

Writing to the Reserved Field: If you write to the reserved field, always write back the field's original value, as described in the note on page 12-5.

Offset I²C1: 0x008
I²C2: 0x108

Access: Mixed

	0	1	2	3	4	5	6	7
R	MEN	MIEN	MSTA	MTX	TXAK		—	BCST
W						RSTA		
Reset	0	0	0	0	0	0	0	0

Figure 12-4. I²C Control Register (I2CCR)

The I2CCR fields are described in Table 12-7.

Table 12-7. I2CCR Field Descriptions

Bits	Name	Description
0	MEN	Module enable. Controls the software reset of the I²C module. 0 The module is reset and disabled. The module is held in reset, but the registers can still be accessed. 1 The I²C module is enabled. MEN must be set before any other control register fields have any effect. All I²C registers for slave receive or master START can be initialized before setting this field, however.
1	MIEN	Module interrupt enable. Enables the interrupt to be reported to the interrupt controller and/or (ultimately) the CPU. 0 Interrupt reporting from the I²C module is disabled. If any pending interrupt conditions exist, they are not cleared. 1 Interrupt reporting from the I²C module is enabled. If an interrupt condition is detected and I2CSR[MIF] is also set, the interrupt is reported.(For more information about I2CSR fields, see Table 12-8.)
2	MSTA	Master/Slave Mode START. 0 If MSTA is changed from 1 to 0, a STOP condition is generated and the I²C mode changes from master to slave. MSTA is cleared without generating a STOP condition when the master loses arbitration. 1 If MSTA is changed from 0 to 1, a START condition is generated on the bus and master mode is selected for the I²C.
3	MTX	Transmit/Receive Mode Select. Specifies the direction of master and slave transfers. If the I²C module is configured as a slave, the software should set MTX to match I2CSR[SRW]. If the I²C module is in master mode, MTX should be set according to the type of transfer required. For address cycles, therefore, this field is always high (has a value of 1). MTX is cleared if the master loses arbitration. 0 Receive mode is selected. 1 Transmit mode is selected.
4	TXAK	Transfer Acknowledge. Specifies the value driven onto the SDA line during acknowledge cycles for both master and slave receivers. The TXAK value applies only if the I²C module is configured as a receiver, not as a transmitter. The TXAK setting 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 the SDA) is sent out to the bus at the ninth clock after receiving 1 byte of data. 1 No acknowledge signal (high value on the SDA) is sent.
5	RSTA	Repeated START. Specifies whether to generate a repeated START condition. Setting RSTA always generates a repeated START condition on the bus and provides the device with the current bus master. The RSTA field is not readable; an attempt to read RSTA returns a 0. 0 No START condition is generated. 1 A repeated START condition is generated.
6	—	Reserved
7	BCST	Broadcast. 0 The broadcast accept capability is disabled. 1 The I²C is enabled to accept broadcast messages at address 0.

12.4.5 I²C Status Register (I2CSR)

The I²C status register (I2CSR), shown in Figure 12-5, is read-only with the exception of the MIF and MAL fields, which can be cleared by software.

Offset I²C1: 0x00C
I²C2: 0x10C

Access: Mixed

	0	1	2	3	4	5	6	7
R	MCF	MAAS	MBB	MAL	BCSTM	SRW	MIF	RXAK
W								
Reset	1	0	0	0	0	0	0	1

Figure 12-5. I²C Status Register (I2CSR)

The I2CSR fields are described in Table 12-8.

Table 12-8. I2CSR Field Descriptions

Bits	Name	Description
0	MCF	Data Transfer. When one byte of data is transferred, MCF is cleared. MCF is set by the falling edge of the ninth clock of a byte transfer. 0 A byte transfer is in progress. MCF is cleared when I2CDR is read in receive mode or written in transmit mode. (For more information about I2CDR, see Table 12-9.) 1 The byte transfer is completed.
1	MAAS	Addressed as a slave. MAAS is set if the module is acting as a slave and has detected that the I2CADR address matches with the transaction's calling address. The processor is interrupted if I2CCR[MIEN] is set. Next, the processor must check the SRW value and set I2CCR[MTX] accordingly. Any write to I2CCR automatically clears MAAS. For more information, see Table 12-5 (I2CADR fields) and Table 12-7 (I2CCR fields). 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 The I²C bus is idle. 1 The I²C bus is busy.
3	MAL	Arbitration Lost. MAL is automatically set (value of 1) if arbitration 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. The broadcast address is always all zeros. BCSTM can be set only if I2CCR[BCST] is set to enable it. (For more information about I2CCR[BCST], see Table 12-7.) 0 There has not been a broadcast match. 1 The calling address matches with the broadcast address instead of the programmed slave address. BCSTM is also set if the I²C drives an address of all zeros and broadcast mode is enabled.
5	SRW	Slave Read/Write. If 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 mode is selected, with the master writing to the slave. 1 Slave transmit mode is selected, with the master reading from the slave. SRW is valid only if both of the following conditions are met: • A complete transfer occurred and no other transfers have been initiated. • The I²C module is configured as a slave and has an address match. By checking SRW, the processor can select slave transmit/receive mode according to the master's command.

Table 12-8. I2CSR Field Descriptions (continued)

Bits	Name	Description
6	MIF	Module Interrupt. MIF is set if an interrupt is detected. An interrupt is reported if I2CCR[MIEN] is set. The interrupts for I²C1 and I²C2 are combined into one interrupt, which is sourced by the dual I²C module.(For more information about I2CCR fields, see Table 12-7.) 0 No interrupt is detected. MIF can be cleared only by software. 1 An interrupt is detected. MIF is set when one of the following events occurs: • One byte of data is transferred (set at the falling edge of the ninth clock). • The I2CADR value matches with the calling address in Slave Receive mode. • Arbitration is lost.
7	RXAK	Received Acknowledge. The value of SDA during the reception of a bus cycle's acknowledge bit. If RXAK = 0, it indicates that an acknowledge signal has been received after the 8 bits of data have been transmitted on the bus. If RXAK = 1, it means no acknowledge signal has been detected at the ninth clock. 0 An acknowledge signal has been received. 1 No acknowledge signal has been received.

12.4.6 I²C Data Register (I2CDR)

I2CDR, shown in Figure 12-6, specifies the calling address and data to be transmitted (if the I²C is in master or slave transmit mode) or allows the I²C module to receive the next byte of data on the I²C module (if the I²C is in master or slave receive mode).

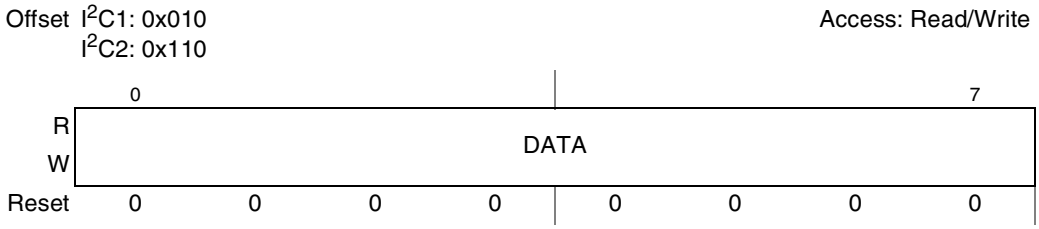


Figure 12-6. I²C Data Register (I2CDR)

The I2CDR field is described in Table 12-9.

Table 12-9. I2CDR Field Descriptions

Bits	Name	Description
0–7	DATA	Specifies data to be transmitted or allows receipt of the next data byte on the I²C module, depending on the I²C mode: • Transmit mode: Data transmission is initiated when data is written to I2CDR. For master transmit mode, the first byte of data written to I2CDR is used for the address transfer and is follows the format described in Section 12.5.2, "Transactions." When bytes are written to I2CDR in transmit mode, they cannot be verified by reading them back. • Receive mode: Reading I2CDR allows the I²C module to receive the next byte of data on the I²C interface (in addition to reading the contents of I2CDR). In all cases, the most significant bit is sent first. I2CCR[MTX] must be set appropriately for the desired behavior. For example, if I2CCR[MTX] is set for transmit mode (1) instead of receive mode (0), reading I2CDR does not initiate receipt of the next data byte. (For more information about I2CCR[MTX], see Table 12-7.) For both master receive and slave receive modes, the very first read is always a dummy read.

12.4.7 Digital Filter Sampling Rate Register (I2CDFSRR)

The digital filter sampling rate register (I2CDFSRR), shown in Figure 12-7, specifies the sample rate for filtering out signal noise.

For additional guidance about the proper use of I2CFDR and I2CDFSRR on Power Architecture™ integrated host/communications processors, refer to the application note AN2919, *Determining the I²C Frequency Divider Ratio for SCL*.

NOTE

Writing to the Reserved Field: If you write to the reserved field, always write back the field's original value, as described in the note on page 12-5.

Offset I²C1: 0x014
I²C2: 0x114

Access: Read/Write

	0	1	2					7
R	—		DFSR					
W								
Reset	0	0	1	0	0	0	0	0

Figure 12-7. I²C Digital Filter Sampling Rate Register (I2CDFSRR)

The I2CDFSRR field is described in Table 12-10.

Table 12-10. I2CDFSRR Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–7	DFSR	Digital Filter Sampling Rate. Specifies the sample rate for the program to use in filtering out signal noise. This rate is used to prescale the frequency the digital filter uses to take samples from the I²C bus. The resulting sampling rate is calculated by dividing the platform(MPX) frequency by the non-zero value of the DFSR. If I2CDFSRR = 0, the I²C bus sample points to the reset divisor.

12.5 Functional Description

This section presents the following topics:

- Section 12.5.1, "Notes About Module Operation"
- Section 12.5.2, "Transactions"
- Section 12.5.3, "Protocol Implementation Details"
- Section 12.5.4, "Bus Arbitration"
- Section 12.5.5, "Clock Behavior"
- Section 12.5.6, "Filtering of SCL and SDA Lines"
- Section 12.5.7, "Boot Sequencer Mode"

12.5.1 Notes About Module Operation

- The I²C module always performs as a slave receiver by default, unless explicitly programmed to be a master or slave transmitter.

- If the device is started in boot sequencer mode (see Section 12.5.7, “Boot Sequencer Mode”), the I²C module will perform as a slave receiver after the boot sequencer mode is completed.
- When the I²C module is acting as a master, it must not try to call its own slave address.

12.5.2 Transactions

This section covers the following topics:

12.5.2.1 Protocol Overview

Figure 12-8 shows the behavior of SCL and SDA during a typical I²C transaction.

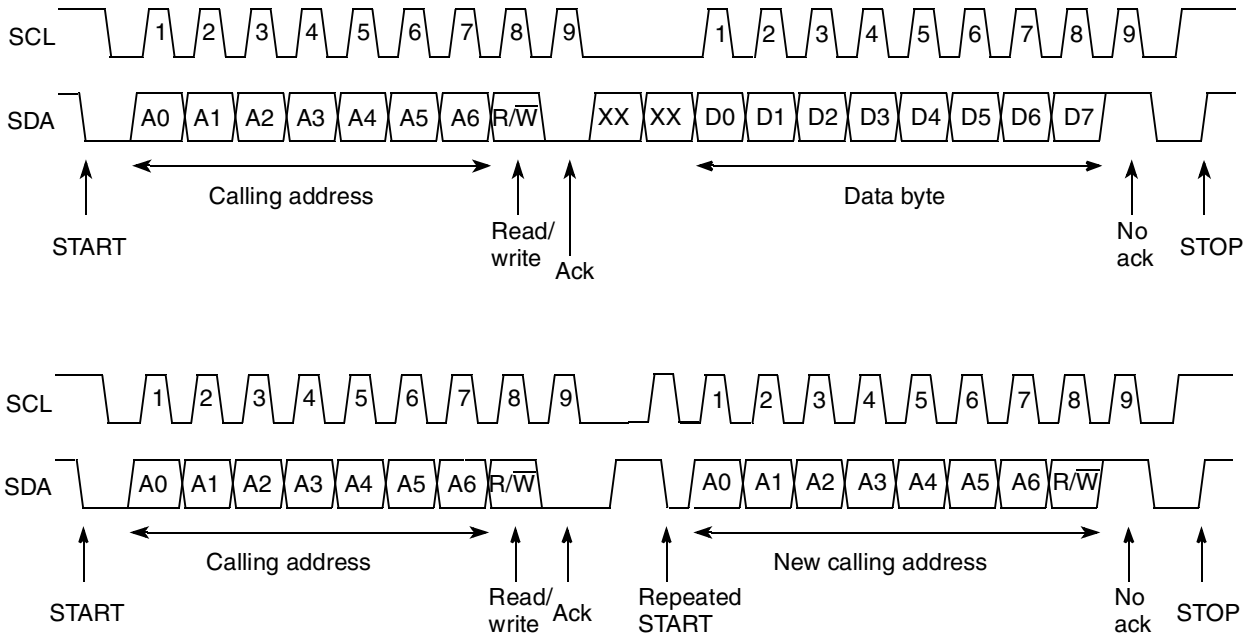


Figure 12-8. I²C Transaction Protocol

12.5.2.2 Definitions

This section defines several important terms presented in Figure 12-8.

Table 12-11. I²C Definitions

Term	Definition
START	A START condition, as defined in Section 12.2.6, "Definition: I²C Conditions"
STOP	A STOP condition, as defined in Section 12.2.6, "Definition: I²C Conditions"
Calling (slave) address	A seven-bit address used to identify a slave on the I²C bus. The requirements for specifying this address are presented in Section 12.5.2.3, "I²C Calling Address Requirements."
Read/write (R/$\overline{W}$)	A bit that specifies the direction of the data transfer to the slave as follows: • 0 = The data is being transferred from the master to the slave ("write") • 1 = The data is being transferred from the slave to the master ("read")
Acknowledge	A bit that specifies the acknowledgement of a calling address, indicated by pulling SDA low.

12.5.2.3 I²C Calling Address Requirements

The calling addresses of the devices used on an I²C network are subject to the following requirements:

- Each slave must have a unique calling address.
- A master must not transmit a calling address that is the same as its own slave address.

12.5.2.4 High-Level Protocol Steps

The I²C protocol conceptually supports two types of transfers, which are illustrated in Figure 12-8. The significant steps in these transfers are presented in Table 12-12. Details of each of these steps are presented in subsequent sections.

Table 12-12. I2C High-Level Protocol Steps

Standard Transfer	Repeated START Transfer
1. START condition 2. Slave target or general call address transmission 3. Data transfer 4. STOP condition 5. (repeat Steps 1–4)	1. START condition 2. Slave target or general call address transmission 3. Data transfer 4. Repeated START condition 5. (repeat Steps 2–4 as needed) 6. STOP condition. 7. (repeat Steps 1–7)

12.5.2.5 START Condition

A master on the I²C bus initiates a data transfer by sending a START condition on the I²C bus when the bus is not engaged (both SDA and SCL are high).

On this device, the START condition is sent by setting I2CCR[MSTA]. (For more information about I2CCR[MSTA], see Table 12-7.)

12.5.2.6 Slave Address Transmission

The master transmits the slave address immediately after the START condition (see Section 12.5.2.5, "START Condition"). The process of slave address transmission is presented in Table 12-13.

Table 12-13. Slave Address Transmission Process

Step	Action
1	The master transmits the seven-bit slave address.
2	The master transmits the R/$\overline{W}$ bit.
3	Each slave examines the transmitted address and compares it to its own. If the addresses match, the slave device returns the acknowledge bit on the ninth SCL clock cycle.
4	The master waits for the acknowledge bit and determines the next step as follows: • The acknowledge bit is set: The master must generate a STOP condition or a repeated START condition. • The acknowledge bit is cleared: The master must wait for SCL to return to logic zero.

12.5.2.7 General Call (Broadcast) Addressing

The master may also initiate a general call (broadcast) command by transmitting a slave address of 0x00. In this case, the second byte of the broadcast message is the master address. This device will not check the R/$\overline{W}$ bit in a broadcast 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 I2CDR with I2CCR[TXAK] = 1, so that it does not interfere with the data written from the addressed device.

This device responds to a general call (broadcast) command if I2CCR[BCST] is set.

12.5.2.8 Data Transmission

A data transfer session has the following characteristics:

- Can transmit one or more bytes of data
- Awakens all slaves
- Proceeds on a byte-by-byte basis in the direction specified by the R/$\overline{W}$ bit sent by the calling master

The transmitted data is subject to the following requirements:

- Each data byte must consist of 8 bits.
- Data bits can be changed only while SCL is low and must be held stable while SCL is high.
- One data bit is transmitted during one SCL clock pulse.
- The most significant bit (msb) must be transmitted first.
- Each data byte must be followed by an acknowledge bit on the ninth SCL clock pulse.

If the slave receiver does not acknowledge the master, the SDA 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 line for the master to generate a STOP or a START condition.

12.5.2.9 STOP Condition

A master on the I^2C bus can terminate a data transfer by sending a STOP condition. It can do so even if the slave has sent an acknowledge bit. In this case, the slave must release the I^2C bus.

On this device, the STOP condition is sent by clearing I2CCR[MSTA]. (For more information about I2CCR[MSTA], see Table 12-7.)

A master is not required to send a STOP condition at the end of every transfer. For more information, see Section 12.5.2.10, “Repeated START Condition.”

12.5.2.10 Repeated START Condition

The I^2C protocol also supports a repeated START condition, which can be generated without a preceding STOP condition. A master device can use this condition to communicate with another slave or with the same slave in a different mode without releasing the bus. This condition is illustrated in the second timing diagram of Figure 12-8.

12.5.3 Protocol Implementation Details

This section provides details of how aspects of the I^2C protocol are implemented in this device. The following sections are included:

- Section 12.5.3.1, “Transaction Monitoring”
- Section 12.5.3.2, “Control Transfer”

12.5.3.1 Transaction Monitoring

The different conditions of the I^2C data transactions (see Section 12.5.2, “Transactions”) are monitored as follows:

- START conditions are detected when SDA falls while SCL is high.
- STOP conditions are detected when SDA rises while SCL 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.

12.5.3.2 Control Transfer

The I²C module contains logic that controls the output to the serial data (SDA) and serial clock (SCL) lines of the I²C. The SCL output is driven low as determined by the internal clock generated in the clock module. The SDA output can only change at the midpoint of a low cycle of the SCL, unless it is performing a START, STOP, or repeated START condition. Otherwise, the SDA output is held constant.

Table 12-14 presents the behavior of the SCL and SDA signals under the various protocol conditions.

Table 12-14. SDA and SCL Signal Behavior

Mode	Conditions When SDA Is Driven Low	Conditions When SCL Corresponds to the Internal SCL Signal
Master	Data bit (transmission) Acknowledge bit (receive) START condition STOP condition Repeated START condition	Bus owner Lost arbitration START condition STOP condition Repeated START condition begin Repeated START condition end
Slave	Acknowledging address match Data bit (transmit) Acknowledge bit (receive)	Address cycle Transmit cycle Acknowledge cycle

12.5.4 Bus Arbitration

This section covers the following topics related to bus arbitration:

- Section 12.5.4.1, “Bus Arbitration Overview”
- Section 12.5.4.2, “Loss of Arbitration”
- Section 12.5.4.3, “Module Startup During a Data Transfer”

12.5.4.1 Bus Arbitration Overview

If two or more masters simultaneously try to control the bus, each master’s clock synchronization procedure (including the I²C module) determines the bus clock. The low part of the period is equal to the longest clock low period and the high part of the period is equal to the shortest one among the masters.

12.5.4.2 Loss of Arbitration

A bus master loses arbitration if it transmits a logic 1 on SDA while another master transmits a logic 0. In this case, the losing master performs the following tasks:

1. Switches to slave-receive mode
2. Stops driving the SDA line without generating a STOP condition
3. Sets I2CSR[MAL] to indicate the loss of arbitration
4. Services the transaction if it is directed to itself

12.5.4.3 Module Startup During a Data Transfer

Table 12-15 presents the behavior of the I²C module if it is enabled in the middle of an ongoing byte transfer.

Table 12-15. Module Behavior at Startup During a Data Transfer

Mode	Behavior
Slave	The I²C module ignores the current transfer on the bus and starts operating whenever a subsequent START condition is detected.
Master	The I²C module cannot tell whether the bus is busy; therefore, if a START condition is initiated, the current bus cycle can be corrupted. This ultimately results in the current bus master of the I²C bus losing arbitration, after which bus operations return to normal.

12.5.5 Clock Behavior

This section covers the following topics of SCL synchronization:

- Section 12.5.5.1, “SCL Synchronization”
- Section 12.5.5.2, “Clock Stretching”
- Section 12.5.5.3, “Handshaking”

12.5.5.1 SCL Synchronization

Due to the wired-AND logic on the SCL line, a high-to-low transition on the SCL line affects all devices connected on the bus. The devices begin counting their low period when the master drives the SCL line low. After a device has driven SCL low, it holds the SCL 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 the SCL line if another device is still within its low period. Therefore, the synchronized clock signal, SCL, 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 period, the synchronized SCL line is released and pulled high. Then there is no difference between the devices’ clocks and the state of the SCL line, and all the devices begin counting their high periods. The first device to complete its high period pulls the SCL line low again.

12.5.5.2 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven the SCL line low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is greater than the master SCL low period, then the resulting SCL bus signal low period is stretched.

12.5.5.3 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfer. Slave devices can hold SCL low after completion of a 1-byte transfer (9 bits). In such cases, the mechanism halts the bus clock and forces the master clock into wait states until the slave releases the SCL line.

12.5.6 Filtering of SCL and SDA Lines

This section covers the following topics:

12.5.6.1 Filtering of SCL and SDA Lines—Overview

The SCL and SDA inputs are filtered to eliminate noise. Three consecutive samples of the SCL and SDA 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.

12.5.6.2 Sample Rate Control

The sampling rate is controlled by the value stored in I2CDFSRR (as described in Table 12-10). The duration of the sampling cycle is controlled by a down counter. This allows a software write to I2CDFSRR to control the filtered sampling rate.

12.5.7 Boot Sequencer Mode

This section covers the following topics

12.5.7.1 Boot Sequencer Mode—Definition

The boot sequencer mode is a special mode of operation that affects the startup of this device. In boot sequencer mode, the device performs the following tasks in order:

1. Holds the processor core in a reset state
2. Reads configuration information from one or more external EEPROMs using the I²C module
3. Transfers this information to the device’s registers or performs external transactions by using the alternate configuration space function
4. Releases the processor core to begin normal operation

12.5.7.2 Selecting Boot Sequencer Mode

Boot sequencer mode is selected at power-on reset (POR) by the settings on the cfg_boot_seq[0:1] reset configuration signals (which are described elsewhere in this document).

12.5.7.3 Boot Sequencer Mode Uses I²C1

The boot sequencer mode uses the first I²C module (I²C1) for communicating with the EEPROM(s). The second module (I²C2) cannot be used for boot sequencer mode.

12.5.7.4 Communication In Boot Sequencer Mode

The following information applies when the I²C module is in boot sequencer mode:

- The I2C1 module accesses the EEPROM at a serial bit clock frequency equal to the platform (MPXCCB) clock frequency divided by 2560.
- No other traffic should be present on the I²C bus.

12.5.7.5 Boot Sequencer EEPROM Data Format

The data in the EEPROM(s) must contain the following information:

- A three-byte preamble
- One or more register preloads (sets of configuration data for loading into the device registers)
- An end command
- A 3-byte cyclic redundancy check (CRC) checksum

Details of the configuration data format are presented in Figure 12-9 and Table 12-16.

<table>
<tr><td>Bit 0</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Bit 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>ACS</td><td colspan="4">BYTE_EN[0:3]</td><td>CONT</td><td colspan="2">ADDR[0:1]</td><td rowspan="7">First register
preload command</td></tr>
<tr><td colspan="8">ADDR[2:9]</td></tr>
<tr><td colspan="8">ADDR[10:17]</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>
</table>

Figure 12-9. Boot Sequencer EEPROM Data Format

ACS	BYTE_EN[0:3]				CONT	ADDR[0:1]		Second register preload command
ADDR[2:9]								
ADDR[10:17]								
DATA[0:7]								
DATA[8:15]								
DATA[16:23]								
DATA[24:31]								
. . .								
ACS	BYTE_EN[0:3]				CONT	ADDR[0:1]		Last register preload command
ADDR[2:9]								
ADDR[10:17]								
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]								Cyclic redundancy check
CRC[8:15]								
CRC[16:23]								
CRC[24:31]								

Figure 12-9. Boot Sequencer EEPROM Data Format

Table 12-16. Boot Sequencer EEPROM Data Format Details

Element	Description
Preamble	A three-byte structure that indicates the beginning of the configuration data. It must have the value 0xAA55AA. The I²C module verifies that the preamble is correctly detected before proceeding further. If a preamble fail occurs: • The device hangs and the external HRESET_REQ signal asserts • The boot sequencer may continue to pull I2C pins low until a hard reset occurs
ACS	Alternate configuration space bit. 0 CCSRBAR is prepended to the EEPROM address. 1 An alternate configuration space address is prepended to the write request from the boot sequencer.

Table 12-16. Boot Sequencer EEPROM Data Format Details (continued)

Element	Description
BYTE_EN	Four bits that control whether the data is written to the configuration register. If a BYTE_EN bit is set, the corresponding DATA byte (as described below) is written to the configuration register. BYTE_EN[0] corresponds to DATA[0:7] BYTE_EN[1] corresponds to DATA[8:15] BYTE_EN[2] corresponds to DATA[16:23] BYTE_EN[3] corresponds to DATA[24:31] **Note:** The values of the BYTE_EN bits should ensure that the data is written contiguously, creating a 1-, 2-, 3-, or 4-byte contiguous write. For example, it is not possible to write only the first and last data bytes (BYTE_EN = 0b1001).
CONT	Continue bit. 0 Do not continue reading data from the EEPROM (there is no more data to be read). In this case, the ACS, BYTE_EN, and ADDR fields must also be cleared. 1 Continue reading data from the EEPROM (there is more data to be read).
ADDR	An 18-bit structure containing the 32-bit (word) address offset. The two low-order bits of the register's byte offset are not included in the boot sequencer command. **Example**: If a register's byte offset is 0xC08, the value of ADDR must be 0x302.
DATA	Four bytes of data. The transfer of these data bytes to their destination registers is controlled by the BYTE_EN bits, as described above.
End command	A 3-byte structure in which every bit is cleared. It indicates the end of the configuration data.
CRC	A four-byte structure containing the cyclic redundancy checksum, which is used to check the integrity of the data. The calculation of the CRC is described in Section 12.5.7.6, "Boot Sequencer Cyclic Redundancy Checksum (CRC) Calculation." If a CRC fail occurs, the device hangs and the external HRESET_REQ signal asserts.

12.5.7.6 Boot Sequencer Cyclic Redundancy Checksum (CRC) Calculation

The module checks the boot sequencer data in the EEPROM using a CRC-32 algorithm with the polynomial shown in Equation 12-1:

Eqn. 12-1

$$1 + x^1 + x^2 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{16} + x^{22} + x^{23} + x^{26} + x^{32}$$

The CRC:

- Is calculated using the above polynomial with a start value of 0xFFFF_FFFF and an XOR with 0x0000_0000
- Must cover all bytes stored in the EEPROM prior to the CRC, including:
  - The preamble
  - All register preloads
  - The end command

12.5.7.7 Boot Sequencer EEPROM Calling Address

When operating in boot sequencer mode, this device uses 0b101_0000 as the EEPROM calling address. The first EEPROM to be addressed must be programmed to respond to this address, or an error is generated.

If more EEPROMs are used, they are addressed in sequential order. Thus, for instance, the calling address of the second EEPROM would be 0b101_0001.

12.5.7.8 Boot Sequencer Addressing Modes

In boot sequencer mode, two addressing modes are possible. These are described in Table 12-17.

Table 12-17. Boot Sequencer Addressing Modes

Addressing Mode	Description
Normal	This mode is used for EEPROM devices with 256 or fewer bytes.
Extended	This mode is used for EEPROM devices with more than 256 bytes. It uses more address bits and is selectable during POR by reconfiguring the cfg_boot_seq[0:1] signals (see their description elsewhere in this document). In this mode, only one EEPROM can be used, and the maximum number of registers is limited by the size of the EEPROM.

12.5.7.9 Communication Sequence In Boot Sequencer Mode

The sequence of communications performed by the I²C1 module is presented in Table 12-18 and depends on the module's addressing mode (see Section 12.5.7.8, "Boot Sequencer Addressing Modes").

Table 12-18. Communication Sequence in Boot Sequencer Mode

Step	Action Taken by the I²C1 Module	
	In Normal Addressing Mode	In Extended Addressing Mode
1	Transmit a reset sequence consisting of a START condition followed by 9 SCL cycles to the EEPROM.	
2	Transmit a reset sequence consisting of a START condition followed by 9 SCL cycles to the EEPROM. **Note:** Steps 1 and 2 clear any transactions that may have been in progress prior to the reset.	
3	Generate a START condition.	
4	Transmit 0b1010_0000 (0xA0). **Note:** This value consists of the calling address of the first target, 0b101_0000 (see Section 12.5.7.7, "Boot Sequencer EEPROM Calling Address"), followed by a write command (the cleared R/$\overline{W}$ bit).	
5	Transmit 0x00. **Note:** This is the 8-bit starting address for the first target.	Transmit 0x00. **Note:** This is the high-order starting address.
6		Transmit 0x00. **Note:** This is the low-order starting address.
7	Generate a repeated START condition.	
8	Transmit 0b1010_0001 (0xA1). **Note:** This value consists of the calling address 0b101_0000 (see Section 12.5.7.7, "Boot Sequencer EEPROM Calling Address") followed by a read command (the set R/$\overline{W}$ bit).	

Table 12-18. Communication Sequence in Boot Sequencer Mode (continued)

Step	Action Taken by the I²C1 Module	
	In Normal Addressing Mode	In Extended Addressing Mode
9	Receive 256 bytes of data from the EEPROM (unless CONT = 0 is encountered; see Section 12.5.7.5, "Boot Sequencer EEPROM Data Format").	Receive data continuously from the EEPROM until: • CONT = 0 is encountered (see Section 12.5.7.5, "Boot Sequencer EEPROM Data Format") • The CRC check is executed (see Section 12.5.7.6, "Boot Sequencer Cyclic Redundancy Checksum (CRC) Calculation") If the last register is not detected (CONT = 0 is never encountered) before wrapping back to the first address, an error condition is detected, causing the device to hang and the HRESET_REQ signal to assert externally.
10	Repeat steps 7–9 with the calling address of the next EEPROM (if present) until: • CONT = 0 is encountered (see Section 12.5.7.5, "Boot Sequencer EEPROM Data Format") • The CRC check is executed (see Section 12.5.7.6, "Boot Sequencer Cyclic Redundancy Checksum (CRC) Calculation") If the last register is not detected (CONT = 0 is never encountered) before wrapping back to the first address, an error condition is detected, causing the device to hang and the HRESET_REQ signal to assert externally.	

12.6 Initialization/Application Information

This section discusses the following topics related to I²C initialization and application:

- Section 12.6.1, "Recommended Interrupt Service Flow"
- Section 12.6.2, "General Programming Guidelines (for Both Master and Slave Mode)"
- Section 12.6.3, "Programming Guidelines Specific to Master Mode"
- Section 12.6.4, "Programming Guidelines Specific to Slave Mode"

12.6.1 Recommended Interrupt Service Flow

Figure 12-10 shows a flowchart for the recommended I²C interrupt service routine. Deviation from the flowchart may result in unpredictable I²C bus behavior. Although the flowchart does not explicitly show

the synchronization after every I2C register access, it is recommended that a synchronizing instruction follow each I²C register read or write to guarantee in-order instruction execution.

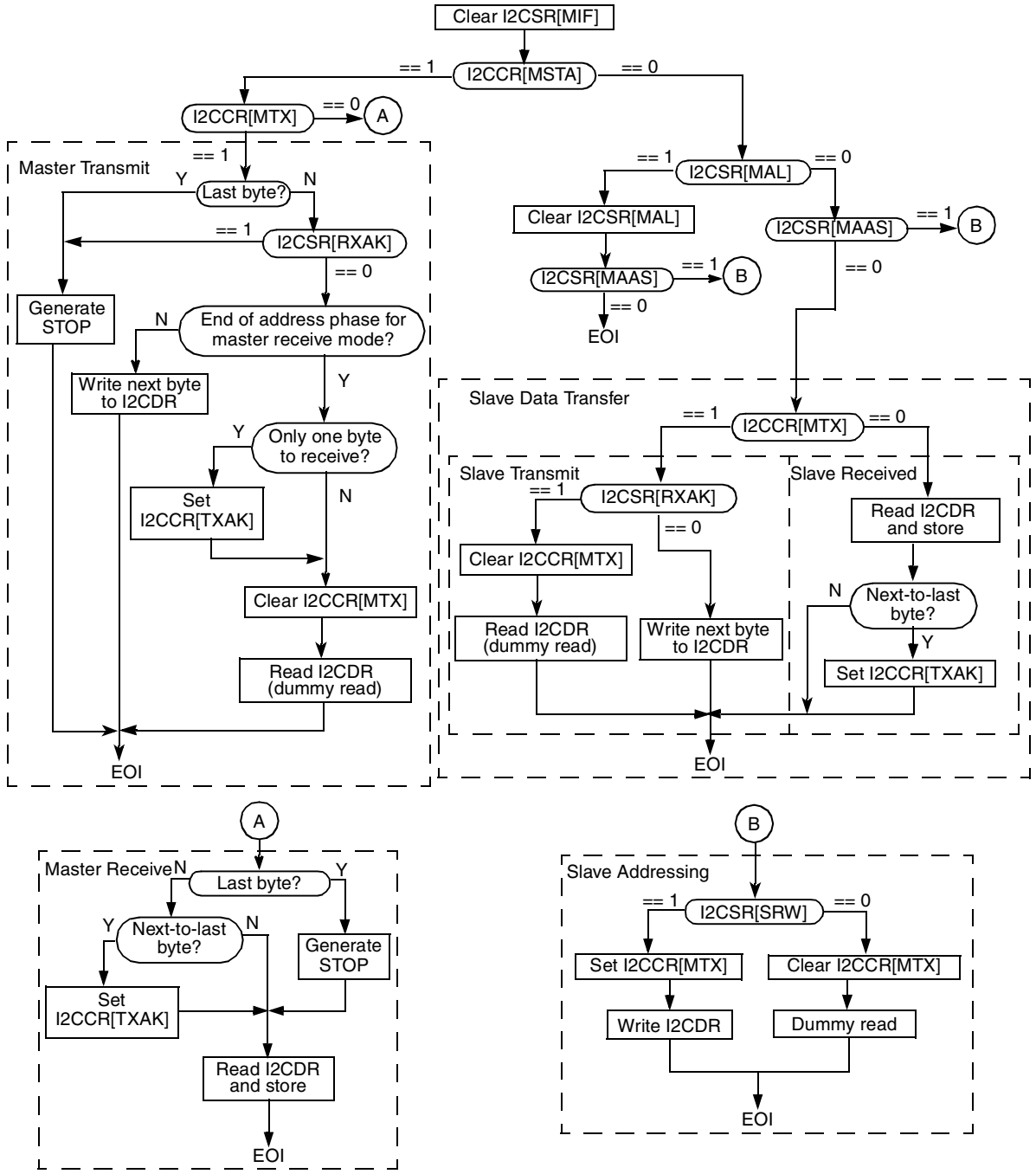


Figure 12-10. Recommended I²C Interrupt Service Routine Flowchart

12.6.2 General Programming Guidelines (for Both Master and Slave Mode)

This section provides programming guidelines recommended for the I²C module in both master and slave mode. It describes the following procedures and processes:

- Section 12.6.2.1, “Initializing the Module”
- Section 12.6.2.2, “Software Response After a Transfer”
- Section 12.6.2.3, “Generating SCL when SDA is Low”

NOTE: Illegal or irregular bus activity

The I²C module does not guarantee its recovery from all illegal I²C bus activity. Additionally, 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 I²C bus hangs. The recovery routine should also handle the case when the status bits returned after an interrupt are not consistent with what was expected due to illegal I²C bus protocol behavior.

12.6.2.1 Initializing the Module

The following sequence initializes the I²C unit:

1. Program I2CFDR[FDR] with the appropriate ratio for the desired SCL frequency. (For more information about the I2CFDR[FDR], see Table 12-6.)
2. Update I2CADR to define the slave address for this device. (For more information about I2CADR fields, see Table 12-5.)
3. Modify I2CCR to select master/slave mode, transmit/receive mode, and interrupt-enable or disable; set I2CCR[MEN] to enable the I²C module. (For more information about I2CCR fields, see Table 12-7.)

12.6.2.2 Software Response After a Transfer

Transmission or reception of a byte automatically sets the data transferring bit (I2CSR[MCF]), which indicates that one byte has been transferred. If the interrupt function is enabled during the initialization sequence (I2CCR[MIEN] is set), the I²C interrupt bit (I2CSR[MIF]) is also set and an interrupt is generated to the PIC. In the interrupt handler, software must take the following steps:

1. Clear I2CSR[MIF].
2. Read the contents of the I²C data register (I2CDR) in receive mode or write to I2CDR in transmit mode. Note that this access to I2CDR causes I2CSR[MCF] to be cleared automatically. See Section 12.6.1, “Recommended Interrupt Service Flow.”

Note the programming guidelines for the following conditions:

- An interrupt at the end of the address cycle—When an interrupt occurs at the end of the address cycle, the master remains in transmit mode. If master receive mode is required, I2CCR[MTX] must be toggled at this stage. See Figure 12-10.
- Monitoring I2CSR[MIF] when the interrupt function is disabled—If the interrupt function is disabled, software can service the I2CDR in the main program by monitoring I2CSR[MIF]. In this

case, I2CSR[MIF] must be polled rather than I2CSR[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 I2CSR[MIF] (or any other I2CSR bits), software delays may be needed in order to give the I²C signals sufficient time to settle.

- Slave-mode addressing—During slave-mode addressing, when I2CSR[MAAS] is set, I2CSR[SRW] should be read to determine the direction of the subsequent transfer, and I2CCR[MTX] should be programmed accordingly.
- Slave-mode data transfers—For slave-mode data transfers (MAAS is cleared), I2CSR[SRW] is not valid, and I2CCR[MTX] must be read to determine the direction of the current transfer. See Figure 12-10 for more details.

12.6.2.3 Generating SCL when SDA is Low

When a system reset does not cause all I²C devices to be reset, it is sometimes necessary to force the I²C module to become the I²C bus master out of reset and drive SCL (even though SDA may already be driven, which indicates the bus is busy). Thus, SDA can be driven low by another I²C device while this I²C module is coming out of reset and will stay low indefinitely.

The following procedure can be used to force this I²C module to generate SCL so that the device driving SDA can finish its transaction:

1. Disable the I²C module (I2CCR[MEN] = 0) and change to master mode (I2CCR[MSTA] = 1) by programming the value 0x20 into I2CCR.
2. Re-enable the I²C module (I2CCR[MEN] = 1) by programming the value 0xA0 into I2CCR.
3. Read I2CDR.
4. Return the I²C module to slave mode (I2CCR[MSTA] = 0) by programming the value 0x80 into I2CCR.

12.6.3 Programming Guidelines Specific to Master Mode

This section includes the following programming guidelines specific to master mode:

12.6.3.1 Generating START

After initialization, the following sequence can be used to generate START:

1. If the device is connected to a multimaster I²C system, test the state of I2CSR[MBB] to check whether the serial bus is available (I2CSR[MBB] = 0) before switching to master mode.
2. Select master mode (set I2CCR[MSTA]) to transmit serial data and select transmit mode (set I2CCR[MTX]) for the address cycle.

3. Write the slave address being called into I2CDR. The data written to I2CDR[0–6] comprises the slave calling address. I2CCR[MTX] indicates the direction of transfer (transmit/receive) required from the slave.

NOTE

The scenario above assumes that the I²C interrupt bit (I2CSR[MIF]) is cleared. If MIF is set at any time, an I²C interrupt is generated (if interrupt reporting is enabled with I2CCR[MIEN] =1).

The interrupts for I²C1 and I²C2 are combined into a single interrupt to the PIC.

12.6.3.2 Generating STOP

A data transfer ends with a STOP condition generated by the master device.

A master transmitter generates a STOP condition after all the data has been transmitted. When the I²C module is acting as the master transmitter, and all the data has been transmitted (that is, after last byte to be transmitted has been written to I2CDR), software clears I2CCR[MSTA] to generate a STOP condition.

When the I²C module is acting as the master receiver, the master indicates the termination of the transfer by not acknowledging the final byte and by generating a STOP condition. For a multiple byte transfer before reading the next-to-last byte in I2CDR, software sets I2CCR[TXAK], then software reads the next-to-last byte in I2CDR, which causes the master receiver to not acknowledge the next transfer and to automatically generate a STOP at the conclusion of the next transfer. For single-byte transfers, at the conclusion of the address phase, software should set I2CCR[TXAK] and clear I2CCR[MTX], and then perform a dummy read to I2CDR. Prior to subsequent I²C transactions, software should clear I2CCR[TXAK]. This can be performed when setting up the I2CCR for the next transfer.

12.6.3.3 Generating Repeated START

At the end of a data transfer, if the master wants to continue communicating 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 I2CCR[RSTA].

12.6.3.4 Loss of Arbitration and Forcing Slave Mode

When a master loses arbitration, the following conditions all occur:

- I2CSR[MAL] is set.
- I2CCR[MSTA] is cleared (changing the master to slave mode).
- An interrupt occurs (if enabled) at the falling edge of the ninth clock of this transfer.

Thus, the slave interrupt service routine should first test I2CSR[MAL] and software should clear it if it is set.

12.6.4 Programming Guidelines Specific to Slave Mode

This section includes the following programming guidelines specific to slave mode:

- Section 12.6.4.1, “Slave Mode Interrupt Service Routine”
- Section 12.6.4.2, “Acknowledge Receipt During Transmission”

12.6.4.1 Slave Mode Interrupt Service Routine

In the slave interrupt service routine, the slave should be tested as follows to determine whether a calling of its own address has been received:

If I2CSR[MAAS] is set, software should set the transmit/receive mode select bit (I2CCR[MTX]) according to the R/$\overline{W}$ command bit (I2CSR[SRW]). Writing to I2CCR clears MAAS automatically. MAAS is read as set only in the interrupt handler at the end of that address cycle in which an address match occurred; interrupts resulting from subsequent data transfers clear MAAS.

A data transfer can then be initiated by writing to I2CDR for slave transmits or dummy reading from I2CDR in slave-receive mode. The slave drives SCL low between byte transfers. SCL is released when the I2CDR is accessed in the required mode. See Figure 12-10.

12.6.4.2 Acknowledge Receipt During Transmission

In the slave transmitter routine, the received acknowledge bit (I2CSR[RXAK]) must be checked before the next byte of data is sent:

- If I2CSR[RXAK] is cleared, the master has acknowledged the previous data transfer and the next byte of data may be transmitted.
- If I2CSR[RXAK] is set, the master is signaling an end-of-data by not acknowledging the previous data transfer. Software running on the slave transmitter must do the following:
  1. Clear I2CCR[MTX] to switch the slave from transmitter to receiver mode.
  2. Perform a dummy read of I2CDR, which releases SCL so the master can generate a STOP condition.

In the slave receiver routine, the receiver transmit acknowledge bit (I2CCR[TXAK]) must be set after the next-to-last byte of data from I2CDR is read.

See Section 12.6.1, “Recommended Interrupt Service Flow.”

Chapter 13
DUART

This chapter describes the dual universal asynchronous receiver/transmitters (DUART). It describes the functional operation, the initialization sequence, and the programming details for the DUART registers and features.

13.1 Overview

The DUART consists of two universal asynchronous receiver/transmitters (UARTs). Note that this device implements two DUART modules DUART1 contains UART0 and UART1; DUART2 contains UART2 and UART3.The UARTs act independently; all references to UART refer to one of these receiver/transmitters. Each UART is clocked by the platform (MPX) 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 13-1, each UART module consists of the following:

- Receive and transmit buffers
- Clear to send ($\overline{\text{CTS}}$) input port and request to send ($\overline{\text{RTS}}$) output port for data flow control
- 16-bit counter for baud rate generation
- Interrupt control logic

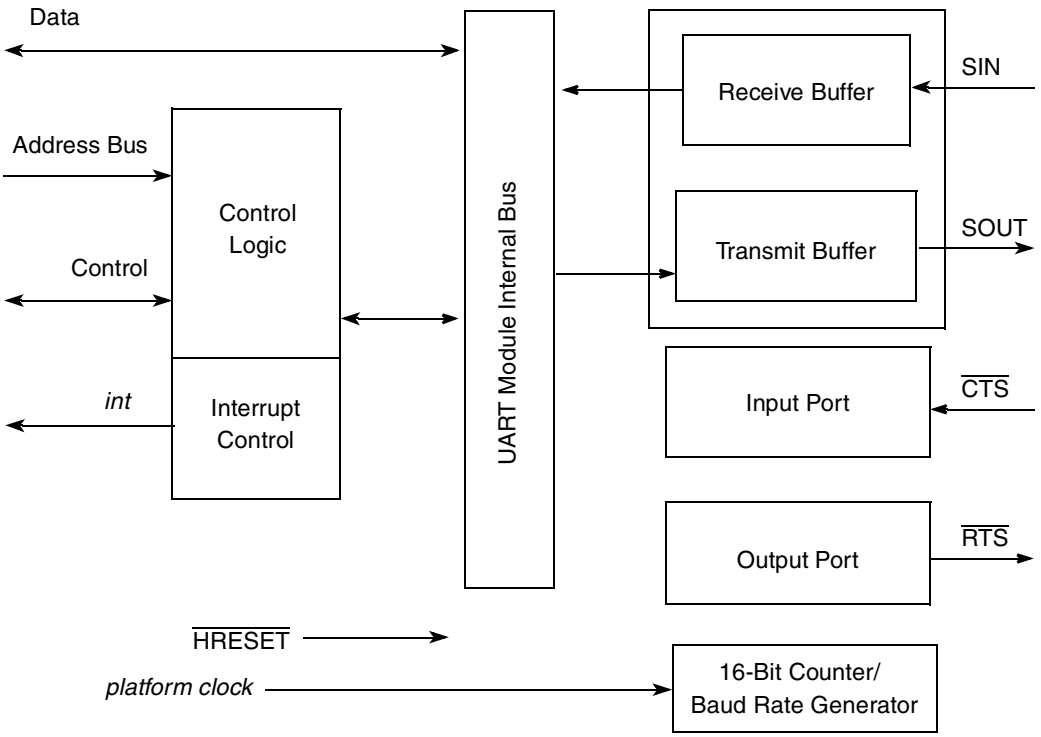


Figure 13-1. UART Block Diagram

13.1.1 Features

The DUART includes these distinctive features:

- Full-duplex operation
- Programming model compatible with original PC16450 UART and PC16550D (improved version of 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 platform clock by 1 to ($2^{16} - 1$) and generate a 16× 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

13.1.2 Modes of Operation

The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the platform 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.

13.1.2.1 DUART Signal Mode Selection

The UART signals are multiplexed with other functions on the device. The following relationships apply:

- UART0 shares signals with GPIO2, SPI, and UART2
- UART1 shares signals with GPIO2, IR2, and UART3
- UART2 shares signals with GPIO2 and UART0
- UART3 shares signals with GPIO2 and UART1

See Section 3.2.5, "UART, SPI, and IR2, and GPIO2 Signal Multiplexing," for more information. The functionality of these signals is determined by the UART0, UART2, UART1, and UART3 fields in the general-purpose I/O control register (GPIOCR) in the global utilities block. Note that the GPIOCR defaults to GPIO functionality on the UART signal pins; the GPIOCR must be initialized to the desired UART signaling for proper UART operation.

Figure 13-2 shows the signal multiplexing for UART0 $\overline{\text{CTS0}}$/$\overline{\text{RTS0}}$ and UART2 SIN2/SOUT2; the multiplexing for UART1 $\overline{\text{CTS1}}$/$\overline{\text{RTS1}}$ and UART3 SIN3/SOUT3 is similar.

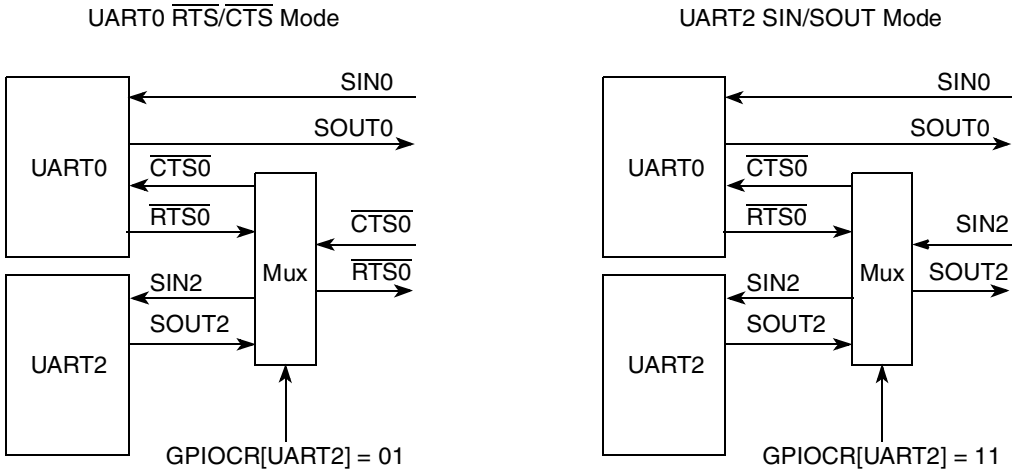


Figure 13-2. UART0/UART2 Signal Multiplexing

13.2 External Signal Descriptions

The DUART signals are described in Table 13-1. 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. Also note that depending on the signal multiplexing selected by GPIOCR, some signals may not be available. See Section 13.1.2.1, "DUART Signal Mode Selection," for more information.

Table 13-1. DUART Signals—Detailed Signal Descriptions

Signal	I/O	Description	
UART_SIN[0:3]	I	Serial data in. Data is received on the receivers of UART0, UART1, UART2, and UART3 through the respective serial data input signal, with the least-significant bit received first. Note that SIN2 is not available when UART0 is configured for $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ mode and SIN3 is not available when UART1 is configured for $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ mode.	
		State Meaning	Asserted/Negated—Represents the data being received on the UART interface.
		Timing	Assertion/Negation—An internal logic sample signal, *rxcnt*, uses the frequency of the baud-rate generator to sample the data on SIN.
UART_SOUT[0:3]	O	Serial data out. The serial data output signals for the UART0, UART1, UART2, and UART3 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. Note that SOUT2 is not available when UART0 is configured for $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ mode and SOUT3 is not available when UART1 is configured for $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ mode.	
		State Meaning	Asserted/Negated—Represents the data being 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.

Table 13-1. DUART Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
UART_CTS[0:1]	I	Clear to send. These active-low inputs are the clear-to-send inputs. They are connected to the respective 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. Note that CTS0 is not available when UART2 is configured for SIN/SOUT mode and CTS1 is not available when UART3 is configured for SIN/SOUT mode.	
		State Meaning	Asserted/Negated—Represent the clear to send condition for their respective UART.
		Timing	Assertion/Negation—Sampled at the rising edge of every platform clock.
UART_RTS[0:1]	O	Request to send. UART_RTSx are active-low output signals that can be programmed to be automatically negated and asserted by either the receiver or transmitter. When connected to the clear-to-send (CTS) input of a transmitter, this signal can be used to control serial data flow. Note that RTS0 is not available when UART2 is configured for SIN/SOUT mode and RTS1 is not available when UART2 is configured for SIN/SOUT mode.	
		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 platform clock.

13.3 Memory Map/Register Definition

Table 13-2 lists the DUART registers and their offsets. It lists the address, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of CCSRBAR together with the block base address and offset listed in Table 13-2.

There are four complete sets of DUART registers (one for each UART). The four UARTs on the device are identical, except that the registers for each UART are located at different offsets. Throughout this chapter, the registers are described by a singular acronym: for example, LCR represents the line control register for either UART0, UART1, UART2, or UART3.

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 13.3.1.8, “Line Control Registers (ULCRn),” for more information on ULCR[DLAB].

All the DUART registers are one byte wide. Reads and writes to these registers must be byte-wide operations. Table 13-2 provides a register summary with references to the section and page that contains detailed information about each register. Undefined byte address spaces within offset 0x000–0xFFF are reserved.

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 13-2. DUART Register Summary

DUART—Block Base Address 0x0_4000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
UART0 Registers				
0x500	URBR—ULCR[DLAB] = 0 UART0 receiver buffer register	R	0x00	13.3.1.1/13-7
0x500	UTHR—ULCR[DLAB] = 0 UART0 transmitter holding register	W	0x00	13.3.1.2/13-7
0x500	UDLB—ULCR[DLAB] = 1 UART0 divisor least significant byte register	R/W	0x00	13.3.1.3/13-8
0x501	UIER—ULCR[DLAB] = 0 UART0 interrupt enable register	R/W	0x00	13.3.1.4/13-9
0x501	UDMB—ULCR[DLAB] = 1 UART0 divisor most significant byte register	R/W	0x00	13.3.1.3/13-8
0x502	UIIR—ULCR[DLAB] = 0 UART0 interrupt ID register	R	0x01	13.3.1.5/13-10
0x502	UFCR—ULCR[DLAB] = 0 UART0 FIFO control register	W	0x00	13.3.1.6/13-12
0x502	UAFR—ULCR[DLAB] = 1 UART0 alternate function register	R/W	0x00	13.3.1.7/13-13
0x503	ULCR—ULCR[DLAB] = x UART0 line control register	R/W	0x00	13.3.1.8/13-14
0x504	UMCR—ULCR[DLAB] = x UART0 modem control register	R/W	0x00	13.3.1.9/13-15
0x505	ULSR—ULCR[DLAB] = x UART0 line status register	R	0x60	13.3.1.10/13-16
0x506	UMSR—ULCR[DLAB] = x UART0 modem status register	R	0x00	13.3.1.11/13-17
0x507	USCR—ULCR[DLAB] = x UART0 scratch register	R/W	0x00	13.3.1.12/13-18
0x510	UDSR—ULCR[DLAB] = x UART0 DMA status register	R	0x01	13.3.1.13/13-19
UART1 Registers				
0x600–0x610	UART1 Registers[1]			
UART2 Registers				
0x700–0x710	UART2 Registers[1]			
UART3 Registers				
0x800–0x810	UART3 Registers[1]			

[1] UART1 has the same memory-mapped registers that are described for UART0 from 0x500 to 0x510, except the offsets range from 0x600 to 0x610. Similarly, the registers for UART2 are located at offsets 0x700 to 0x710 and the registers for UART3 are located at offsets 0x800 to 0x810.

13.3.1 Register Descriptions

The following sections describe the UART*n* registers.

13.3.1.1 Receiver Buffer Registers (URBR*n*) (ULCR[DLAB] = 0)

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[RXRDY] 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 13.3.1.10, "Line Status Registers (ULSRn)." Figure 13-4 shows the receiver buffer registers. Note that these registers have same offset as the UTHRs.

Figure 13-3 shows the bits in the URBRs.

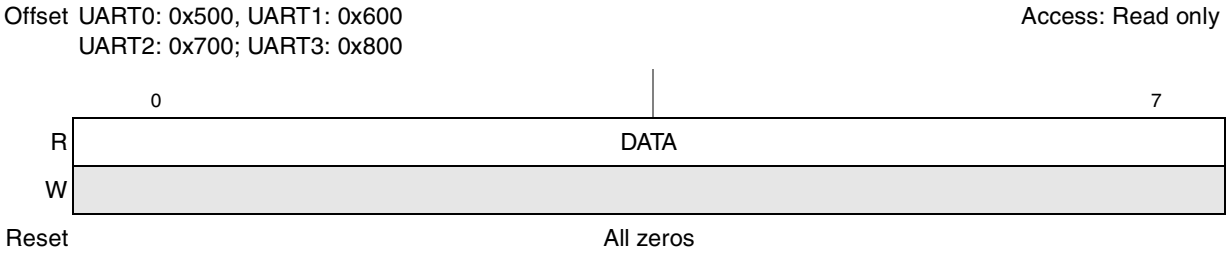


Figure 13-3. Receiver Buffer Registers (URBR*n*)

Table 13-3 describes the fields of URBR.

Table 13-3. URBR Field Descriptions

Bits	Name	Description
0–7	DATA	Data received from the transmitter on the UART bus (read only)

13.3.1.2 Transmitter Holding Registers (UTHR*n*) (ULCR[DLAB] = 0)

A write to these 8-bit registers causes the UART devices to transfer 5–8 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[$\overline{\text{TXRDY}}$] indicates when the FIFO is full. Refer to Table 13-20 and Table 13-21 for more details.

Figure 13-4 shows the bits in the UTHRs.

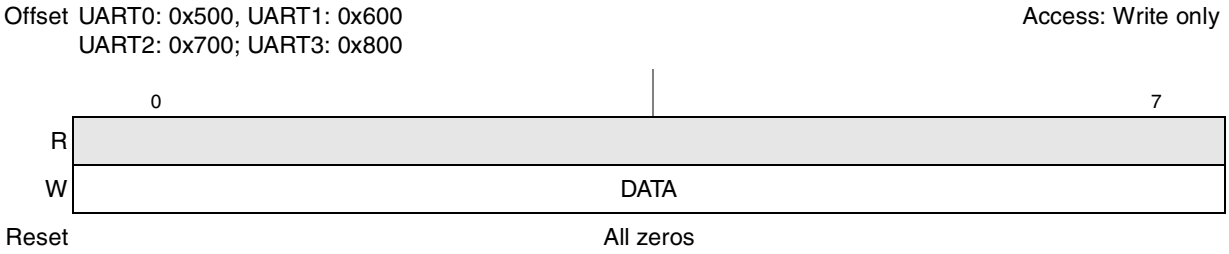


Figure 13-4. Transmitter Holding Registers (UTHR*n*)

Table 13-4 describes the fields of UTHR.

Table 13-4. UTHR Field Descriptions

Bits	Name	Description
0–7	DATA	Data that is written to UTHR (write only)

13.3.1.3 Divisor Most and Least Significant Byte Registers (UDMB and UDLB) (ULCR[DLAB] = 1)

The divisor least significant byte register (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 13-7.

Figure 13-5 shows the bits in the UDMBs.

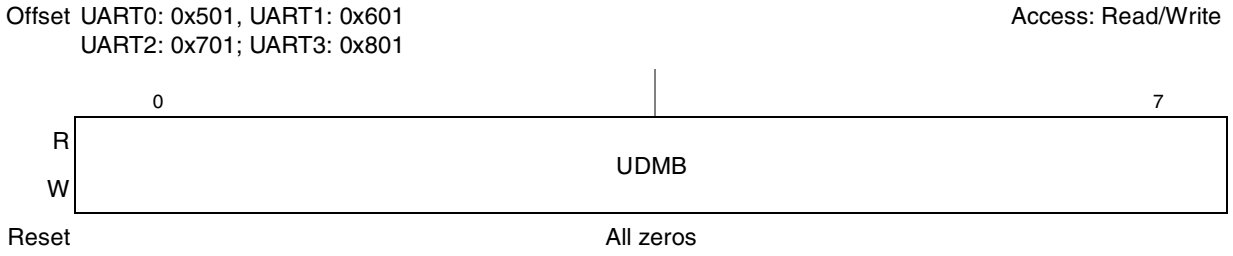


Figure 13-5. Divisor Most Significant Byte Registers (UDMB0, UDMB1)

Table 13-5 describes the fields of UDMB registers.

Table 13-5. UDMB Field Descriptions

Bits	Name	Description
0–7	UDMB	Divisor most significant byte

Figure 13-6 shows the bits in the UDLBs.

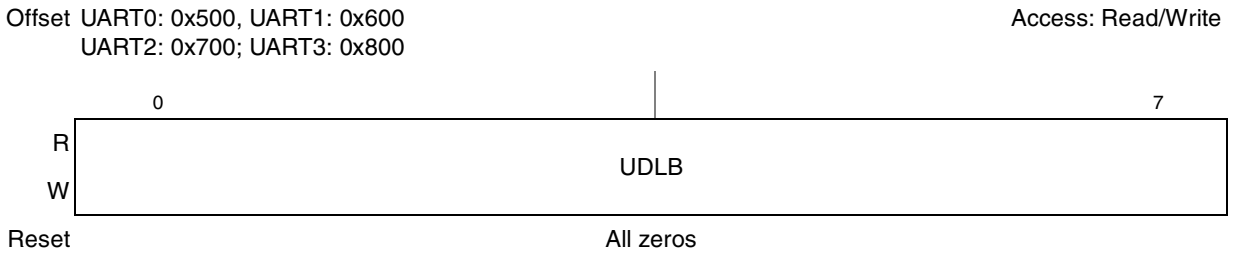


Figure 13-6. Divisor Least Significant Byte Registers (UDLB*n*)

Table 13-6 describes the fields of UDLB registers.

Table 13-6. UDLB Field Descriptions

Bits	Name	Description
0–7	UDLB	Divisor least significant byte. This is concatenated with UDMB.

Table 13-7 shows examples of baud rate generation based on common input clock frequencies. Many other target baud rates are also possible. Note that because only integer values can be used as divisors, the actual baud rate differs slightly from the desired (target) baud rate; for this reason, both target and actual baud rates are given, along with the percentage of error.

Table 13-7. Baud Rate Examples

Target Baud Rate (Decimal)	Divisor		Platform Clock (MPX) Frequency (MHz)	Actual Baud Rate (Decimal)	Percent Error (Decimal)
	Decimal	Hex			
9,600	1736	6C8	266	9600.61444	0.0064
19,200	868	364	266	19,201.22888	0.0064
38,400	434	1B2	266	38,402.45776	0.0064
57,600	289	121	266	57,670.12673	0.1217
115,200	145	91	266	114,942.52844	0.2235
230,400	72	48	266	231,481.48090	0.4694
9,600	2170	87A	333	9600.61444	0.0064
19,200	1085	43D	333	19,201.22888	0.0064
38,400	543	21F	333	38,367.09638	0.0858
57,600	362	16A	333	57,550.64457	0.0857
115,200	181	B5	333	115,101.28913	0.0857
230,400	90	5A	333	231,481.48148	0.4694
9,600	4340	10F4	667	9600.61443	0.0064
19,200	2170	87A	667	19,201.22886	0.0064
38,400	1085	43D	667	38,402.45772	0.0064
57,600	723	2D3	667	57,630.24429	0.0525
115,200	362	16A	667	115,101.28902	0.0857
230,400	181	B5	667	230,202.57804	0.0857

13.3.1.4 Interrupt Enable Register (UIER) (ULCR[DLAB] = 0)

The UIER gives the user the ability to mask specific UART interrupts to the programmable interrupt controller (PIC).

Figure 13-7 shows the bits in the UIER.

Offset UART0: 0x501, UART1: 0x601
UART2: 0x701; UART3: 0x8010

Access: Read/Write

	0–3	4	5	6	7
R W	—	EMSI	ERLSI	ETHREI	ERDAI
Reset	All zeros				

Figure 13-7. Interrupt Enable Register (UIER)

Table 13-8 describes the fields of UIER.

Table 13-8. 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 the clear-to-send bit in the UART modem status register (UMSR) 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 the FIFO mode

13.3.1.5 Interrupt ID Registers (UIIR*n*) (ULCR[DLAB] = 0)

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:

1. Receiver line status
2. Received data ready/character time-out
3. Transmitter holding register empty
4. Modem status

See Table 13-10 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 13-8 shows the bits in the UIIR.

Offset UART0: 0x502, UART1: 0x602
UART2: 0x702; UART3: 0x802

Access: 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 13-8. Interrupt ID Registers (UIIR)

Table 13-9 describes the fields of the UIIR.

Table 13-9. 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 13-10. IID3 is set along with IID2 only when a timeout interrupt is pending for FIFO mode.
5–6	IID2–1	Interrupt ID bits identify the highest priority interrupt that is pending as indicated in Table 13-10.
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 13-10.

Table 13-10. UIIR IID Bits Summary

IID Bits IID[3–0]	Priority Level	Interrupt Type	Interrupt Description	How To Reset Interrupt
0b0001	—	—	—	—
0b0110	Highest	Receiver line status	Overrun error, parity error, framing error, or break interrupt	Read the line status register.
0b0100	Second	Received data available	Receiver data available or trigger level reached in FIFO mode	Read the receiver buffer register or interrupt is automatically reset if the number of bytes in the receiver FIFO drops below the trigger level.
0b1100	Second	Character time-out	No characters have been removed from or input to the receiver FIFO during the last 4 character times and there is at least one character in the receiver FIFO during this time.	Read the receiver buffer register.

Table 13-10. UIIR IID Bits Summary (continued)

IID Bits IID[3–0]	Priority Level	Interrupt Type	Interrupt Description	How To Reset Interrupt
0b0010	Third	UTHR empty	Transmitter holding register is empty	Read the UIIR or write to the UTHR.
0b0000	Fourth	Modem status	$\overline{\text{CTS}}$ input value changed since last read of UMSR	Read the UMSR.

13.3.1.6 FIFO Control Registers (UFCR*n*) (ULCR[DLAB] = 0)

The UFCR, a write-only register, 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.

When the UFCR bits are written, the FIFO enable bit must also be set or else the UFCR bits are not programmed. When changing from FIFO mode to 16450 mode (non-FIFO mode) and vice versa, data is automatically cleared from the FIFOs.

After all 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 bits.

Figure 13-9 shows the bits in the UFCRs.

Offset UART0: 0x502, UART1: 0x602
UART2: 0x702; UART3: 0x802

Access: Write only

	0	1	2	3	4	5	6	7
R			—					
W	RTL				DMS	TFR	RFR	FEN
Reset	All zeros							

Figure 13-9. FIFO Control Registers (UFCR*n*)

Table 13-11 describes the fields of the UFCRs.

Table 13-11. 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 the designated interrupt trigger level as follows: 00 1 byte 01 4 bytes 10 8 bytes 11 14 bytes
2–3	—	Reserved
4	DMS	DMA mode select. See Section 13.4.5.2, “DMA Mode Select,” for more information. 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.

Table 13-11. UFCR Field Descriptions (continued)

Bits	Name	Description
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 Enables the transmitter and receiver FIFOs

13.3.1.7 Alternate Function Registers (UAFR*n*) (ULCR[DLAB] = 1)

The UAFRs give software the ability to gate off the baud clock and write to both UART0/UART1 registers or both UART2/UART3 registers simultaneously with the same write operation.

Figure 13-10 shows the bits in the UAFRs.

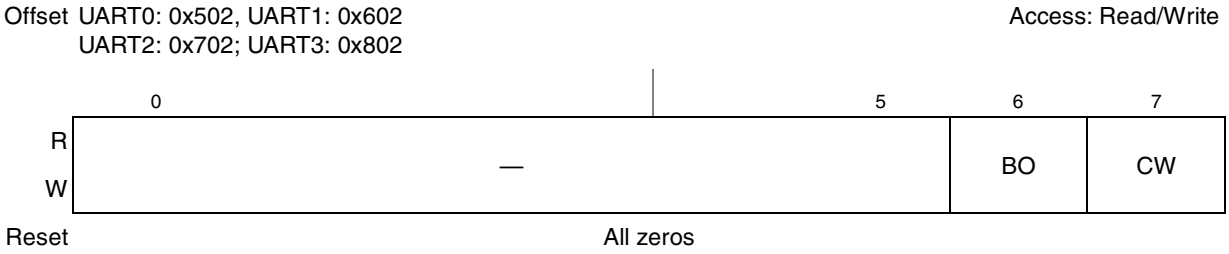


Figure 13-10. Alternate Function Register (UAFR)

Table 13-12 describes the fields of the UAFRs.

Table 13-12. 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 UART0 and UART1 (DUART1) or UART2 and UART3 (DUART2) 1 Enables concurrent writes to corresponding UART registers. For DUART1, a write to a register in UART0 is also a write to the corresponding register in UART1 and vice versa; for DUART2, a write to a register in UART2 is also a write to the corresponding register in UART3 and vice versa. The user needs to ensure that the LCR[DLAB] of both UARTs are in the same state before executing a concurrent write to register addresses 0x*n*00, 0x*n*01 and 0x*n*02, where *n* is the offset of the corresponding UART.

13.3.1.8 Line Control Registers (ULCR*n*)

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 the ULCR, the software should not re-write the ULCR when valid transfers on the UART bus are active. The software should not re-write the ULCR until the last STOP bit has been received and there are no new characters 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 13-14 for more information. ULCR[NSTB], defines the number of STOP bits to be sent at the end of the data transfer. The receiver only checks 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 that are transmitted or received as a serial character. The word length does not include START, parity, and STOP bits.

Figure 13-11 shows the bits in the ULCRs.

Offset UART0: 0x503, UART1 0x603
UART2: 0x703; UART3: 0x803

Access: Read/Write

	0	1	2	3	4	5	6	7
R W	DLAB	SB	SP	EPS	PEN	NSTB	WLS	
Reset	All zeros							

Figure 13-11. Line Control Register (ULCR)

Table 13-13 describes the fields of the ULCRs.

Table 13-13. 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 divisor latch least and most significant byte registers and alternate function register (UAFR)
1	SB	Set break. 0 Send normal UTHR data onto the serial output (SOUT) signal 1 Force logic 0 to be on the SOUT signal. 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. And if PEN = 1 and EPS = 0, mark parity is selected.
3	EPS	Even parity select. See Table 13-14 for more information. 0 If PEN = 1 and SP = 0, odd parity is selected. 1 If PEN = 1 and SP = 0, even parity is selected.

Table 13-13. ULCR Field Descriptions (continued)

Bits	Name	Description
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, 1½ 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. The word length select values are as follows: 00 5 bits 01 6 bits 10 7 bits 11 8 bits

Table 13-14. 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
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

13.3.1.9 Modem Control Registers (UMCR*n*)

The UMCRs control the interface with the external peripheral device on the UART bus.

Figure 13-12 shows the bits in the UMCRs.

Offset UART0: 0x504, UART1: 0x604
UART2: 0x704; UART3: 0x804

Access: Read/Write

	0	2	3	4	5	6	7
R W	—		LOOP	—		RTS	—
Reset	All zeros						

Figure 13-12. Modem Control Register (UMCR)

Table 13-15 describes the fields of UMCRs.

.

Table 13-15. UMCR Field Descriptions

Bits	Name	Description
0–2	—	Reserved.
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.

13.3.1.10 Line Status Registers (ULSR*n*)

The ULSRs are read-only registers that 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.

Figure 13-13 shows the bits in the ULSRs.

Offset UART0: 0x505, UART1: 0x605
UART2: 0x705; UART3: 0x805

Access: 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 13-13. Line Status Register (ULSR)

Table 13-16 describes the fields of the ULSRs.

Table 13-16. ULSR Field Descriptions

Bits	Name	Description
0	RFE	Receiver FIFO error. 0 This bit is 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 to one 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 The 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 This bit is 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 This bit is 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, this bit 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 receives the following new data.
5	PE	Parity error. 0 This bit is cleared when ULSR is read or when a new character is loaded into the URBR. 1 Unexpected parity value encountered when receiving data. In FIFO mode, the character with the error is at the top of the FIFO.
6	OE	Overrun error. 0 This bit is cleared when ULSR is read. 1 Before the URBR is read, the URBR was overwritten with a new character. The old character is loss. 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 This bit is cleared when URBR is read or when all of the data in the receiver FIFO is read. 1 A character has been received in the URBR or the receiver FIFO.

13.3.1.11 Modem Status Registers (UMSR*n*)

The UMSRs track the status of the modem (or external peripheral device) clear to send ($\overline{\text{CTS}}$) signal for the corresponding UART.

Figure 13-14 shows the bits in the UMSRs.

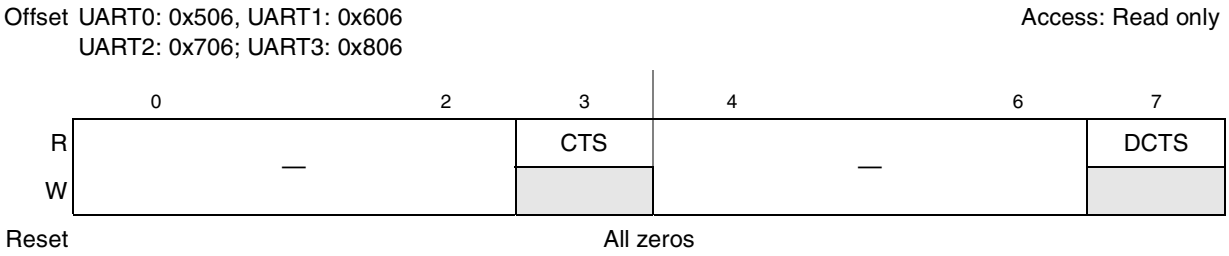


Figure 13-14. Modem Status Register (UMSR)

Table 13-17 describes the fields of the UMSRs.

Table 13-17. UMSR Field Descriptions

Bits	Name	Description
0–2	—	Reserved.
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.
7	DCTS	Clear to send. 0 No change on the corresponding $\overline{\text{CTS}}n$ signal since the last read of UMSR[CTS] 1 The $\overline{\text{CTS}}n$ value has changed, since the last read of UMSR[CTS]. Causes an interrupt if UIER[EMSI] is set to detect this condition

13.3.1.12 Scratch Registers (USCR*n*)

The USCR registers are for debugging software or the DUART hardware. The USCRs do not affect the operation of the DUART.

Figure 13-15 shows the bits in USCRs.

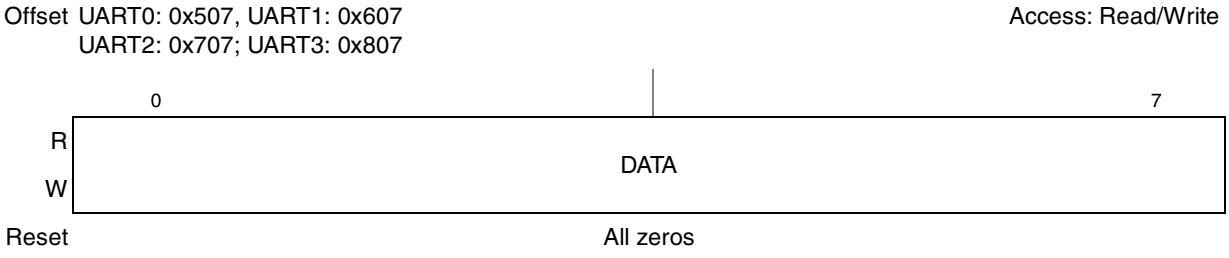


Figure 13-15. Scratch Register (USCR)

Table 13-18 describes the fields of the USCRs.

Table 13-18. USCR Field Descriptions

Bits	Name	Description
0–7	DATA	Data

13.3.1.13 DMA Status Registers (UDSR*n*)

The DMA status registers (UDSRs) are read-only registers that return transmitter and receiver FIFO status. UDSRs also provide the ability to assist DMA data operations to and from the FIFOs.

Figure 13-16 shows the bits in UDSRs.

Offset UART0: 0x510, UART1: 0x610
UART2: 0x710; UART3: 0x810

Access: Read only

	0				5	6	7
R	—					TXRDY	RXRDY
W							
Reset	0	0	0	0	0	0	0	1

Figure 13-16. DMA Status Register (UDSR)

Table 13-19 describes the fields of the UDSRs.

Table 13-19. UDSR Field Descriptions

Bits	Name	Description
0–5	—	Reserved
6	TXRDY	Transmitter ready. This read-only bit reflects the status of the transmitter FIFO or the UTHR. The status depends on the DMA mode selected, which is determined by the DMS and FEN bits in the UFCR. 0 The bit is cleared, as shown in Table 13-21. 1 This bit is set, as shown in Table 13-20.
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 the DMS and FEN bits in the UFCR. 0 The bit is cleared, as shown in Table 13-23. 1 This bit is set, as shown in Table 13-22.

Table 13-20. 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 13-21. 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 when the transmitter FIFO is not yet full.

Table 13-22. 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 13-23. UDSR[RXRDY] Cleared Conditions

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.

13.4 Functional Description

The communication channel provides a full-duplex asynchronous receiver and transmitter using an operating frequency derived from the platform clock signal.

The transmitter accepts parallel data with a write access to the transmitter holding register (UTHR). In FIFO mode, the data is placed directly into an internal transmitter shift register, or into the transmitter FIFO—see Section 13.4.5, “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.

13.4.1 Serial Interface

The UART bus is a serial, full-duplex, point-to-point bus as shown in Figure 13-17. Therefore, only two devices are attached to the same signals and there is no need for address or arbitration bus cycles.

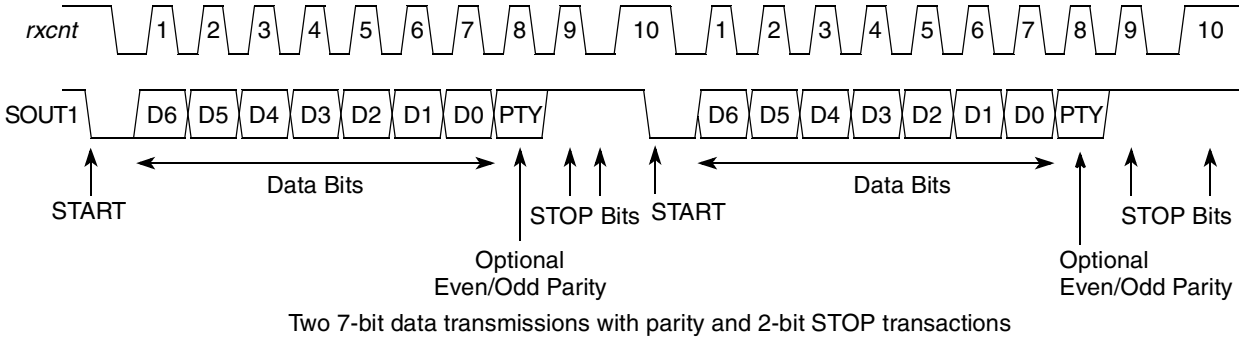


Figure 13-17. UART Bus Interface Transaction Protocol Example

A standard UART bus transfer is composed of either three or four parts:

- START bit
- Data transfer bits (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.

13.4.1.1 START Bit

A write to the transmitter holding register (UTHR) generates a START bit on the SOUT signal. Figure 13-17 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 the UART line control register (ULCR). When the bus is idle, SOUT is high.

13.4.1.2 Data Transfer

Each data transfer contains 5–8 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 5–8 of the data bits previously written to the UTHR. The data bits are driven from the least significant to the most significant bits. After the parity and STOP bits, a new data transfer can begin if new data is written to the UTHR.

13.4.1.3 Parity Bit

The user has the option of using even, odd, no parity, or stick parity (see Section 13.3.1.8, “Line Control Registers (ULCRn).” Both the receiver and transmitter parity definition must agree before attempting to transfer data. When receiving data a parity error can occur if an unexpected parity value is detected. (See Section 13.3.1.10, “Line Status Registers (ULSRn).”)

13.4.1.4 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.

13.4.2 Baud-Rate Generator Logic

Each UART contains an independent programmable baud-rate generator, that is capable of taking the platform 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:

$$\text{Baud rate} = (1/16) \times (\text{platform clock frequency} \div \text{divisor value})$$

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 the UAFR[BO] bit. This can be used to determine baud rate errors.

13.4.3 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).

13.4.4 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.

13.4.4.1 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 re-synchronize 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.

13.4.4.2 Parity Error

A parity error occurs, and ULSR[PE] is set, when unexpected parity values are encountered while receiving data. 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.

13.4.4.3 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.

13.4.5 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. The DMA status registers (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.

13.4.5.1 FIFO Interrupts

In FIFO mode, the UIER[ERDAI] is set when a time-out interrupt occurs. When a receive data time-out occurs there is a maskable interrupt condition (through UIER[ERDAI]). See Section 13.3.1.4, “Interrupt Enable Register (UIER) (ULCR[DLAB] = 0),” for more details on interrupt enables.

The interrupt ID register (UIIR) indicates if the FIFOs are enabled. Interrupt ID3 UIIR[IID3] bit is only set 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 there is at least one character in the receiver FIFO during this time. The character time-out interrupt (controlled by UIIR[IID*n*]) is cleared when the URBR is read. See Section 13.3.1.5, “Interrupt ID Registers (UIIRn) (ULCR[DLAB] = 0),” for more information.

The UIIR[FE] bits indicate if FIFO mode is enabled.

13.4.5.2 DMA Mode Select

The UDSR[RXRDY] bit reflects the status of the receiver FIFO or URBR. In mode 0 (UFCR[DMS] is cleared), UDSR[RXRDY] is cleared when there is at least one character in the receiver FIFO or URBR and it is set when there are no more characters in the receiver FIFO or URBR. This occurs regardless of the setting of the UFCR[FEN] bit. 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 and it is set when there are no more characters in the receiver FIFO.

The UDSR[TXRDY] bit 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 and it is set after the first character is loaded into the transmitter FIFO or UTHR. This occurs regardless of the setting of the UFCR[FEN] bit. 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 and it is set when the transmitter FIFO is full.

See Section 13.3.1.13, “DMA Status Registers (UDSRn),” for a complete description of the USDR[RXRDY] and USDR[TXRDY] bits.

13.4.5.3 Interrupt Control Logic

An interrupt is active when DUART interrupt ID register bit 7 (UIIR[IID0]), is cleared. The interrupt enable register (UIER) is used to mask specific interrupt types. For more details refer to the description of UIER in Section 13.3.1.4, “Interrupt Enable Register (UIER) (ULCR[DLAB] = 0).”

When the interrupts are disabled in UIER, polling software cannot use UIIR[IID0] to determine whether the UART is ready for service. The software must monitor the appropriate bits in the line status (ULSR) and/or the modem status (UMSR) registers. UIIR[IID0] can be used for polling if the interrupts are enabled in UIER.

13.5 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-wide 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 14
Fast/Serial Infrared Interfaces (FIRI/SIRI)

The two IrDA-compliant fast/serial infrared interfaces (FIRI/SIRI) support high-, medium-, and low-speed infrared communications through external circuitry that converts received infrared signals to electrical signals or transforms electrical signals to signals that drive an infrared LED for transmission. These controllers support the following speeds:

- High speed—4 Mbits/s (FIRI)
- Medium speed—0.576 and 1.152 Mbits/s (FIRI)
- Low speed—9.6–115.2 Kbits/s (SIRI)

When the device comes out of reset, the FIRI/SIRI controllers are configured to send and receive at the lowest speed, 9.6 Kbps. (This is an IrDA standard.) The receive signal (IR*n*_TXD) is forwarded to both FIRI and SIRI modules; the transmit signal (IR*n*_TXD) is multiplexed, and IRCR[IR*n*SEL] determines which signal is presented on the pin. Both controllers share an external clock input that is used only for fast infrared; the serial infrared module is clocked internally (frequency is MPX clock ÷ 2).

Communication using the infrared interfaces usually involves DMA requests. The DMA transfers data to and from FIFOs within the controller, which is set to generate DMA requests when the receive FIFO is nearly full or the transmit FIFO is nearly empty. The FIRI module has two independent 128-byte FIFOs for transmitter and receiver. The SIRI module has a 32-byte send FIFO and a 32-half-word receive FIFO.

Each module has a separate set of registers. The SIRI registers are located at 0x2_D000 and 0x2_E000; the FIRI registers at 0x2_D100 and 0x2_E100.

14.1 Introduction

The SIRI module, based on traditional UART technology, transmits and receives characters containing either 7 or 8 bits (program selectable). Data is written to a 32-byte transmit FIFO (TxFIFO). This data is passed to a shift register and shifted serially out on the transmitter pin (TXD). Data is received serially from the receiver pin (RXD) and stored in a 32-halfword-deep receive FIFO (RxFIFO). The received data is retrieved from the RxFIFO. The FIFOs generate maskable interrupts as well as DMA requests when the data level in one of the FIFOs reaches a programmed threshold level.

The SIRI generates baud rates based on a dedicated input clock (MPX clock ÷ 2) and its programmable divisor. The SIRI also contains programmable auto-baud detection circuitry to receive one or two stop bits as well as odd, even, or no parity. The receiver detects framing errors, idle conditions, BREAK characters, parity errors, and overrun errors.

The FIRI module is capable of establishing a 0.576-Mbit/s, 1.152-Mbit/s or 4-Mbit/s half-duplex link. It supports 0.576-Mbit/s and 1.152-Mbit/s medium infrared (MIR) physical layer protocol and 4-Mbit/s fast

infrared (FIR) physical layer protocol defined by IrDA version 1.4. Figure 14-1 shows how the LED and IR detector are shared for both modules and can be controlled via IRCR[IR*n*SEL].

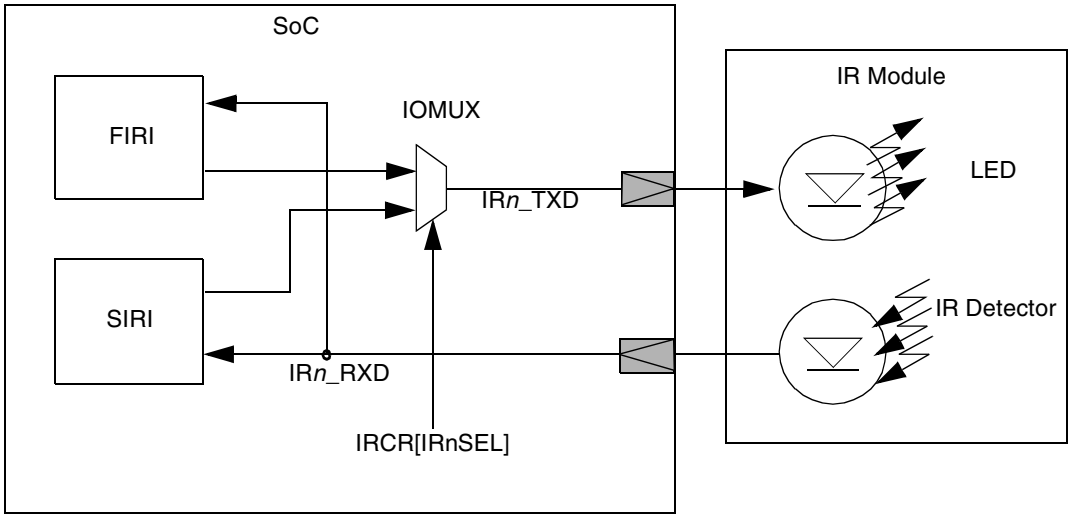


Figure 14-1. FIRI Simplified Block Diagram

14.2 Features

The SIRI includes the following features:

- Low-speed IrDA-compatible infrared interface (up to 115.2 Kbit/s)
- 7 or 8 data bits
- 1 or 2 stop bits
- Two independent 32-entry FIFOs for transmit and receive
- Programmable parity (even, odd, or no parity)
- Status flags for various flow control and FIFO states
- Voting logic for improved noise immunity (16x oversampling)
- Transmitter FIFO empty interrupt suppression
- SIRI internal clocks enable/disable
- Auto baud rate detection (up to 115.2 Kbit/s)
- Receiver and transmitter enable/disable for power saving
- Maskable interrupts
- Two DMA requests (TxFIFO DMA request and RxFIFO DMA request)
- Escape character sequence detection
- Software reset ($\overline{\text{SRST}}$)

The FIRI includes the following features:

- Protocol support:
  — 0.576 Mbit/s MIR

  - 1.152 Mbit/s MIR
  - 4 Mbit/s FIR (4PPM)
- Device destination detection hardware support
- Interrupt generation
- DMA capability
- SIP generation for collision avoidance

14.2.1 Modes of Operation

The FIRI supports the following modes:

- Hardware packet assembly
  - FIR mode
  - 0.576-Mbps MIR mode
  - 1.152-Mbps MIR mode
  - Serial infrared interaction pulse (SIP) generation
- Hardware packet search
  - FIR mode
  - 0.576 Mbps MIR mode
  - 1.152 Mbps MIR mode

14.3 External Signal Descriptions

The FIRI signals are described in detail in Table 14-1.

NOTE

The IR*n* signals are multiplexed with GPIO2 signals. See Section 3.2.4, "IR1 and GPIO2 Signal Multiplexing," and Section 3.2.5, "UART, SPI, and IR2, and GPIO2 Signal Multiplexing," for more information. The functionality of these signals is determined by the general-purpose I/O control register (GPIOCR) in the global utilities block. Note that the GPIOCR defaults to GPIO functionality on the IR*n* signal pins; the GPIOCR must be initialized to the desired IR*n* signaling for proper operation.

Table 14-1. FIRI Signals—Detailed Signal Descriptions

Signal	I/O	Description	
IR*n*_RXD	I	Receive data. Data is received on the receivers of FIRI1 and FIRI2.	
		State Meaning	Asserted/Negated—Represents the data being received on the FIRI*n* interface.
		Timing	Assertion/Negation—

Table 14-1. FIRI Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
IR*n*_TXD	O	Transmit data. The serial data output signals for FIRI1 and FIRI2..	
		State Meaning	Asserted/Negated—Represents the data being transmitted on the FIRI*n* interface.
		Timing	Assertion/Negation—
IR_CLKIN	I	FIRI clock input. Note that the SIRI input clock, referred to in this chapter as *siri_clkin*, is MPX clock ÷ 2.	
		State Meaning	
		Timing	

14.4 Memory Map/Register Definition

Table 14-2 lists the FIRI/SIRI registers and their offsets. It lists the address, name, and a cross-reference to the complete description of each register. Note that the full register address is comprised of CCSRBAR together with the block base address and offset listed in Table 14-2.

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.

Note that all FIRI/SIRI registers can be accessed as 32-, 16-, or 8-bit quantities. For example, STXD[TX_DATA] can be the rightmost 8 bits of a 32-bit access to offset 0x040, the lower 8 bits of a 16-bit access to 0x042, or as simply an 8-bit access to 0x043.

Table 14-2. FIRI/SIRI Register Summary

Offset	Register	Access	Reset	Section/Page
Block Base Address: 0x2_D000				
0x000	SIRI receiver register (SRXD)	R	All zeros	14.4.1.1/14-5
0x040	SIRI transmitter register (STXD)	W	All zeros	14.4.1.2/14-6
0x080	SIRI control register 1 (SCR1)	R/W	All zeros	14.4.1.3/14-7
0x084	SIRI control register 2 (SCR2)	R/W	0x0000_0001	14.4.1.4/14-9
0x088	SIRI control register 3 (SCR3)	R/W	0x0000_0700	14.4.1.5/14-11
0x08C	SIRI control register 4 (SCR4)	R/W	0x0000_8000	14.4.1.6/14-12
0x090	SIRI FIFO control register (SFCR)	R/W	0x0000_0801	14.4.1.7/14-14
0x094	SIRI status register 1 (SSR1)	Mixed	0x0000_2040	14.4.1.8/14-15

Table 14-2. FIRI/SIRI Register Summary (continued)

Offset	Register	Access	Reset	Section/Page
0x098	SIRI status register 2 (SSR2)	Mixed	0x0000_4448	14.4.1.9/14-17
0x09C	SIRI escape character register (SESC)	R/W	0x0000_002B	14.4.1.10/14-18
0x0A0	SIRI escape timer register (STIM)	R/W	All zeros	14.4.1.11/14-19
0x0A4	SIRI BRM incremental register (SBIR)	R/W	All zeros	14.4.1.12/14-19
0x0A8	SIRI BRM modulator register (SBMR)	R/W	All zeros	14.4.1.13/14-20
0x0AC	SIRI baud rate count register (SBRC)	R	0x0000_0004	14.4.1.14/14-20
0x0B0	SIRI one millisecond register (SONEMS)	R/W	All zeros	14.4.1.15/14-21
0x0B4	SIRI test register (STS)	R/W	0x0000_0060	14.4.1.16/14-21
0x100	FIRI transmit control register (FIRITCR)	Mixed	All zeros	14.4.1.17/14-23
0x104	FIRI transmit count register (FIRITCTR)	R/W	All zeros	14.4.1.18/14-25
0x108	FIRI receive control register (FIRIRCR)	R/W	All zeros	14.4.1.19/14-26
0x10C	FIRI transmit status register (FIRITSR)	Mixed	All zeros	14.4.1.20/14-28
0x110	FIRI receive status register (FIRIRSR)	Mixed	All zeros	14.4.1.21/14-29
0x114	Transmitter FIFO	W	All zeros	—
0x118	Receiver FIFO	R/W	All zeros	—
0x11C	FIRI control register (FIRICR)	R/W	All zeros	14.4.1.22/14-30
Block Base Address: 0x2_E000				
The second FIRI/SIRI block contains identical registers as the first, with the same offsets. The base address, however, differs as shown above.				

14.4.1 Register Descriptions

This section describes the FIRI and SIRI registers in detail.

14.4.1.1 SIRI Receiver Register (SRXD)

Figure 14-2 shows the SIRI receiver register.

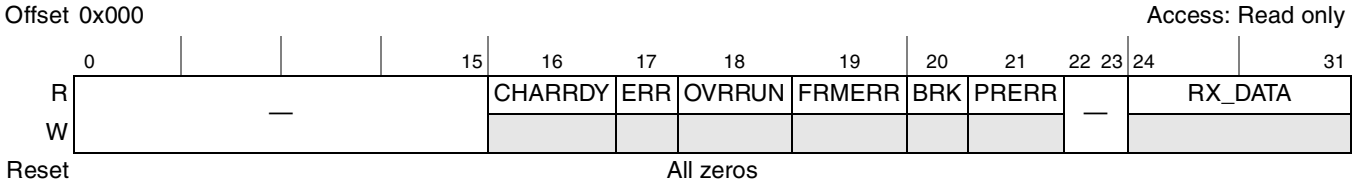


Figure 14-2. SIRI Receiver Register (SRXD)

NOTE

Memory space between SRXD and STXD registers (for example, BASE + 0x04 up to BASE + 0x3c) cannot be accessed. Any read or write access to this space is considered an invalid address and results in a transfer error.

Table 14-3 shows SRXD field descriptions.

Table 14-3. SRXD Field Descriptions

Bits	Field	Description
0–15	—	Reserved
16	CHARRDY	Character ready. This read-only bit indicates an invalid read when the FIFO becomes empty and the software tries to read the same old data. This bit should not be used for polling for data written to the RxFIFO. 0 Character in RX_DATA field and associated flags are invalid. 1 Character in RX_DATA field and associated flags valid and ready for reading.
17	ERR	Error detect. Indicates whether the character present in the RX_DATA field has an error (OVRRUN, FRMERR, BRK or PRERR) status. The ERR bit is updated and valid for each received character. 0 No error status was detected. 1 An error status was detected.
18	OVRRUN	Receiver overrun. This read-only bit indicates when set that the corresponding character was stored in the last position (32nd) of the RxFIFO. Even if a 33rd character has not been detected, this bit is set to one for the 32nd character. 0 No RxFIFO overrun was detected. 1 A RxFIFO overrun was detected.
19	FRMERR	Frame error. Indicates whether the current character had a framing error (a missing stop bit) and is possibly corrupted. FRMERR is updated for each character read from the RxFIFO. 0 The current character has no framing error. 1 The current character has a framing error.
20	BRK	BREAK detect. Indicates whether the current character was detected as a BREAK character. The data bits and the stop bit are all 0. The FRMERR bit is set when BRK is set. When odd parity is selected, PRERR is also set when BRK is set. BRK is valid for each character read from the RxFIFO. 0 The current character is not a BREAK character. 1 The current character is a BREAK character.
21	PRERR	Parity error. Indicates if the current character was detected with a parity error and is possibly corrupted. PRERR is updated for each character read from the RxFIFO. When parity is disabled, PRERR always reads as 0. 0 = No parity error was detected for data in the RX_DATA field. 1 = A parity error was detected for data in the RX_DATA field.
22–23	—	Reserved
24–31	RX_DATA	Received data. Holds the received character. In 7-bit mode, the most significant bit (MSB) is forced to 0. In 8-bit mode, all bits are active.

14.4.1.2 SIRI Transmitter Register (STXD)

Figure 14-3 shows the SIRI transmitter register.

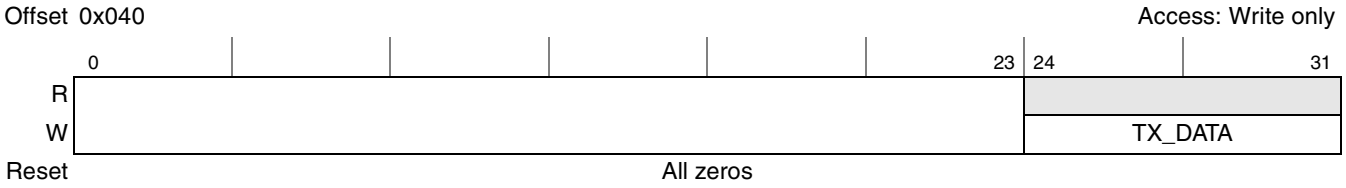


Figure 14-3. SIRI Transmitter Register (STXD)

NOTE

Memory space between SRXD and STXD registers (for example, BASE + 0x04 up to BASE + 0x3c) cannot be accessed. Any read or write access to this space is considered an invalid address and results in a transfer error.

Table 14-4 shows the STXD field descriptions.

Table 14-4. STXD Field Descriptions

Bits	Field	Description
0–23	—	Reserved
24–31	TX_DATA	Transmit data. Holds the parallel transmit data inputs. In 7-bit mode, D7 is ignored. In 8-bit mode, all bits are used. Data is transmitted least significant bit (LSB) first. A new character is transmitted when the TX_DATA field is written. The TX_DATA field must be written only when the TRDY bit is high to ensure that corrupted data is not sent.

14.4.1.3 SIRI Control Register 1 (SCR1)

Figure 14-4 shows the SIRI control register 1.

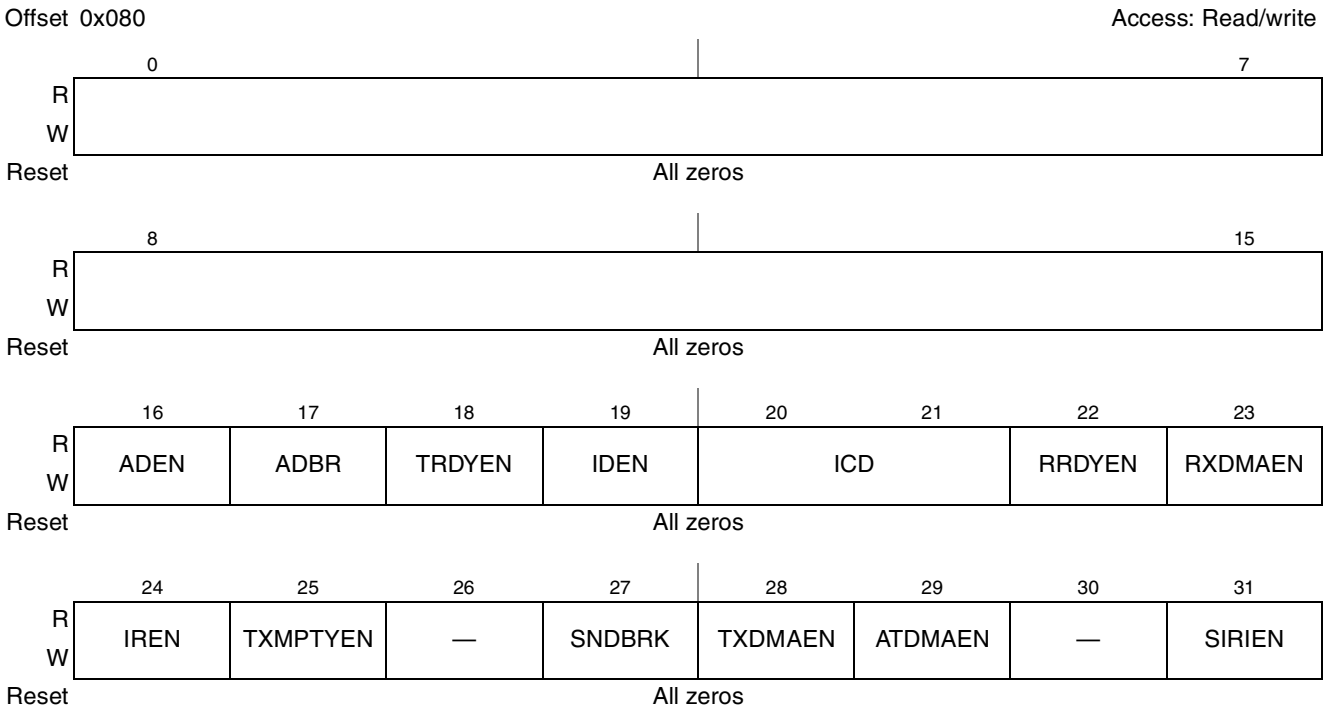


Figure 14-4. SIRI Control Register 1 (SCR1)

Table 14-5 shows the SCR1 field descriptions.

Table 14-5. SCR1 Field Descriptions

Bits	Field	Description
0–15	—	Reserved
16	ADEN	Automatic baud rate detection interrupt enable. Enables/disables the automatic baud rate detect complete (ADET) bit to generate an interrupt. 0 Disable the automatic baud rate detection interrupt. 1 Enable the automatic baud rate detection interrupt.
17	ADBR	Automatic detection of baud rate. Enables/disables automatic baud rate detection. When the ADBR bit is set and the ADET bit is cleared, the receiver detects the incoming baud rate automatically. The ADET flag is set when the receiver verifies that the incoming baud rate is detected properly by detecting an ASCII character “A” or “a” (0x61 or 0x41). 0 Disable automatic baud rate detection. 1 Enable automatic baud rate detection.
18	TRDYEN	Transmitter ready interrupt enable. Enables/disables the transmitter ready interrupt (TRDY) when the transmitter has one or more slots available in the TxFIFO. The fill level in the TxFIFO at which an interrupt is generated is controlled by TxTL bits. When TRDYEN is negated, the transmitter ready interrupt is disabled. 0 Disable the transmitter ready interrupt. 1 Enable the transmitter ready interrupt.
19	IDEN	Idle condition detected interrupt enable. Enables/disables the IDLE bit to generate an interrupt. 0 Disable the IDLE bit. 1 Enable the IDLE bit.
20–21	ICD	Idle condition detect. Controls the number of frames RXD is allowed to be idle before an idle condition is reported. 00 Report idle of more than 4 frames. 01 Report idle of more than 8 frames. 10 Report idle of more than 16 frames. 11 Report idle of more than 32 frames.
22	RRDYEN	Receiver ready interrupt enable. Enables/disables the RRDY interrupt when the RxFIFO contains data. The fill level in the RxFIFO at which an interrupt is generated is controlled by the RXTL bits. When RRDYEN is negated, the receiver ready interrupt is disabled. 0 Disable the RRDY interrupt. 1 Enable the RRDY interrupt.
23	RXDMAEN	Receive ready DMA enable. Enables/disables the receive DMA request when the receiver has data in the RxFIFO. The fill level in the RxFIFO at which a DMA request is generated is controlled by the RXFL bits. When negated, the receive DMA request is disabled. 0 Disable the DMA request. 1 Enable the DMA request.
24	IREN	Infrared interface enable. Enables/disables the IR interface. Refer to the IR interface description in Section 14.5.2, “SIRI Operation” for more information. 0 Disable the IR interface. 1 Enable the IR interface.
25	TXMPTYEN	Transmitter empty interrupt enable. Enables/disables the transmitter FIFO empty (TXFE) interrupt. When negated, the TXFE interrupt is disabled. 0 Disable the transmitter FIFO empty interrupt. 1 Enable the transmitter FIFO empty interrupt.
26	—	Reserved

Table 14-5. SCR1 Field Descriptions (continued)

Bits	Field	Description
27	SNDBRK	Send BREAK. Forces the transmitter to send a BREAK character. The transmitter finishes sending the character in progress (if any) and sends BREAK characters until SNDBRK is reset. Because the transmitter samples SNDBRK after every bit is transmitted, it is important that SNDBRK is asserted high for a sufficient period of time to generate a valid BREAK. After the BREAK transmission completes, the SIRI transmits 2 mark bits. The user can continue to fill the TxFIFO. Any characters remaining are transmitted when the BREAK is terminated. 0 Do not send a BREAK character. 1 Send a BREAK character (continuous zeros).
28	TXDMAEN	Transmitter ready DMA enable. Enables/disables the transmit DMA request when the transmitter has one or more slots available in the TxFIFO. The fill level in the TxFIFO that generates the request is controlled by the TXTL bits. 0 Disable the transmit DMA request. 1 Enable the transmit DMA request.
29	ATDMAEN	Aging DMA timer enable. Enables/disables the receive DMA request for the aging timer interrupt (triggered with AGTIM flag in SSR1[8]). 0 Disable the DMA AGTIM interrupt. 1 Enable the DMA AGTIM interrupt.
30	—	Reserved
31	SIRIEN	SIRI Enable. Enables/disables the SIRI. If SIRIEN is negated in the middle of a transmission, the transmitter stops and pulls the TXD line to a logic 1. SIRIEN must be set to 1 before any access to STXD and SRXD registers; otherwise, an error is returned. 0 Disable the SIRI. 1 Enable the SIRI.

14.4.1.4 SIRI Control Register 2 (SCR2)

Figure 14-5 shows the SIRI control register 2.

Offset 0x084 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	ESCI	STXEN	—		ESCEN	—		PREN	PROE	STPB	WS	—	ATEN	TXEN	RXEN	SRST
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 14-5. SIRI Control Register 2 (SCR2)

Table 14-6 shows the field descriptions of SCR2.

Table 14-6. SCR2 Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	ESCI	Escape sequence interrupt enable. Enables/disables the ESCF bit to generate an interrupt. 0 Disable the escape sequence interrupt. 1 Enable the escape sequence interrupt.
17	STXEN	SIRI transmitter enable. This bit must be set to enable transmitting with the SIRI. 0 SIRI transmitter disabled. 1 SIRI transmitter enabled.
18–19	—	Reserved
20	ESCEN	Escape enable. Enables/disables the escape sequence detection logic. 0 Disable escape sequence detection. 1 Enable escape sequence detection.
21–22	—	Reserved
23	PREN	Parity enable. Enables/disables the parity generator in the transmitter and parity checker in the receiver. When PREN is asserted, the parity generator and checker are enabled, and disabled when PREN is negated. 0 Disable the parity generator and checker. 1 Enable the parity generator and checker.
24	PROE	Parity odd/even. Controls the sense of the parity generator and checker. When PROE is high, odd parity is generated and expected. When PROE is low, even parity is generated and expected. PROE has no function if PREN is low. 0 Even parity 1 Odd parity
25	STPB	Stop. Controls the number of stop bits transmitted after a character. When STPB is high, 2 stop bits are sent. When STPB is low, 1 stop bit is sent. STPB has no effect on the receiver, which expects 1 or more stop bits. 0 Transmit 1 stop bit. 1 Transmit 2 stop bits.
26	WS	Word size. Controls the character length. When WS is high, the transmitter and receiver are in 8-bit mode. When WS is low, they are in 7-bit mode. The transmitter ignores bit 7 and the receiver sets bit 7 to 0. WS can be changed in-between transmission (reception) of characters, but not when a transmission (reception) is in progress. During transmission (reception), the length of the current character being transmitted (received) is unpredictable. 0 7-bit transmit and receive character length (not including start, stop or parity bits) 1 8-bit transmit and receive character length (not including start, stop or parity bits)
27	—	Reserved
28	ATEN	Aging timer enable. This bit is used to enable the aging timer interrupt (triggered with AGTIM). 0 Disable the AGTIM interrupt. 1 Enable the AGTIM interrupt.
29	TXEN	Transmitter enable. Enables/disables the transmitter. When TXEN is negated, the transmitter is disabled and idle. When the SIRIEN and TXEN bits are set, the transmitter is enabled. If TXEN is negated in the middle of a transmission, the SIRI disables the transmitter immediately and starts marking ones. The transmitter FIFO cannot be written to when this bit is cleared. 0 Disable the transmitter. 1 Enable the transmitter.

Table 14-6. SCR2 Field Descriptions (continued)

Bits	Name	Description
30	RXEN	Receiver enable. Enables/disables the receiver. When the receiver is enabled, if the RXD input is already low, the receiver does not recognize BREAK characters. The receiver requires a valid 1-to-0 transition before it can accept any character. 0 Disable the receiver. 1 Enable the receiver.
31	SRST	Software reset. Resets the transmitter and receiver state machines, all FIFOs, status registers SSR1 and SSR2, and BRM registers SBIR and SBMR. Once the software clears $\overline{\text{SRST}}$, the software reset remains active for 4 clock cycles of *siri_clkin* before hardware deasserts $\overline{\text{SRST}}$. Software can write only zero to $\overline{\text{SRST}}$; writing one to $\overline{\text{SRST}}$ is ignored. 0 Reset the transmit and receive state machines, all FIFOs, and all status registers. 1 No reset

14.4.1.5 SIRI Control Register 3 (SCR3)

Figure 14-6 shows the SIRI control register 3.

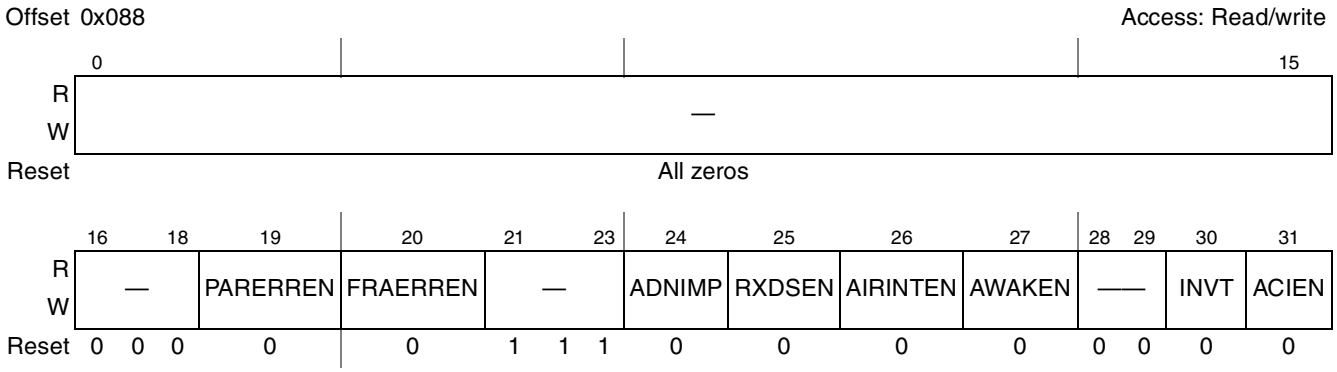


Figure 14-6. SIRI Control Register 3 (SCR3)

Table 14-7 shows the SCR3 field descriptions.

Table 14-7. SCR3 Field Descriptions

Bits	Name	Description
0–18	—	Reserved
19	PARERREN	Parity error interrupt enable. Enables/disables the interrupt. When asserted, PARERREN causes the PARITYERR bit to generate an interrupt. 0 Disable the parity error interrupt. 1 Enable the parity error interrupt.
20	FRAERREN	Frame error interrupt enable. Enables/disables the interrupt. When asserted, FRAERREN causes the FRAMERR bit to generate an interrupt. 0 Disable the frame error interrupt. 1 Enable the frame error interrupt.
21–23	—	Reserved. Reset value is 0b111; do not change.

Table 14-7. SCR3 Field Descriptions (continued)

Bits	Name	Description
24	ADNIMP	Autobaud detection not improved. Disables new features of autobaud detection. Refer to Section 14.5.2.9.2, "Baud Rate Automatic Detection Protocol Improved" for more details. 0 Select autobaud detection new features. 1 Keep old autobaud detection mechanism.
25	RXDSEN	Receive status interrupt enable. Controls the receive status interrupt. When this bit is enabled and RXDS status bit is set, an interrupt is generated. 0 Disable the RXDS interrupt. 1 Enable the RXDS interrupt.
26	AIRINTEN	Asynchronous IR WAKE interrupt enable. Controls the asynchronous IR WAKE interrupt. When AIRINTEN is asserted and a pulse is detected on the SIRI_RX pin, an interrupt is generated. 0 Disable the AIRINT interrupt. 1 Enable the AIRINT interrupt.
27	AWAKEN	Asynchronous WAKE interrupt enable. Controls the asynchronous WAKE interrupt. When AWAKEN is asserted and a falling edge is detected on the RXD pin, an interrupt is generated. 0 Disable the AWAKE interrupt. 1 Enable the AWAKE interrupt.
28–29	—	Reserved. Reset value is 0b00; do not change.
30	INVT	Inverted infrared transmission. Sets the active level for the transmission. When INVT is cleared, the infrared logic block transmits a positive IR 3/16 pulse for all zeros and zeros are transmitted for ones. When INVT is set (INVT = 1), the infrared logic block transmits an active low or negative infrared 3/16 pulse for all zeros and ones are transmitted for ones. 0 Active low transmission 1 Active high transmission.
31	ACIEN	Autobaud counter interrupt enable. This bit is used to enable the autobaud counter stopped interrupt (triggered with ACST (SSR2[11]). 0 Disable ACST interrupt. 1 Enable ACST interrupt

14.4.1.6 SIRI Control Register 4 (SCR4)

Figure 14-7 shows the SIRI control register 4.

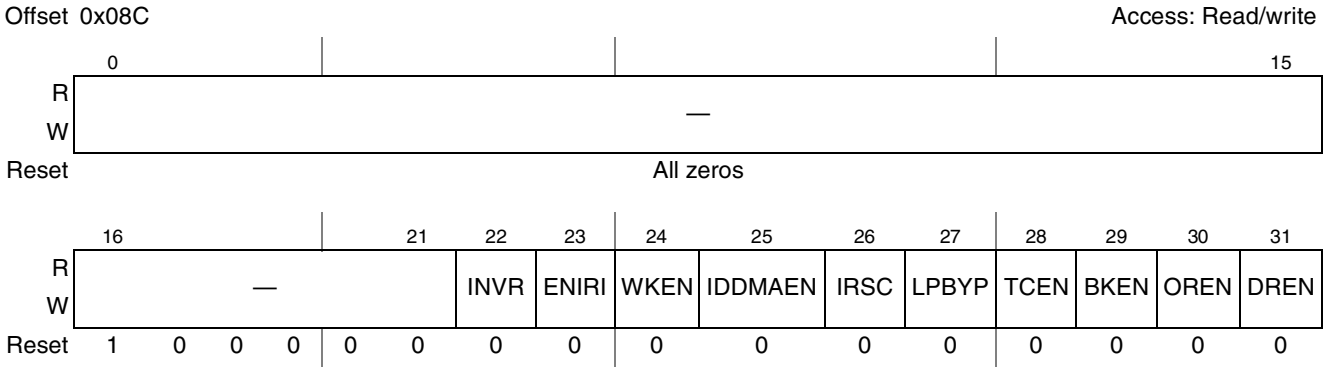


Figure 14-7. SIRI Control Register 4 (SCR4)

Table 14-8 shows the SCR4 field descriptions.

Table 14-8. SCR4 Field Descriptions

Bits	Name	Description
0–21	—	Reserved
22	INVR	Inverted infrared reception. Determines the logic level for the detection. When cleared, the infrared logic block expects an active low or negative IR 3/16 pulse for zeros and ones are expected for ones. When INVR is set (INVR1), the infrared logic block expects an active high or positive IR 3/16 pulse for zeros and zeros are expected for ones. 0 Detect active low. 1 Detect active high.
23	ENIRI	Serial infrared interrupt enable. Enables/disables the serial infrared interrupt. 0 Disable the serial infrared interrupt 1 Enable the serial infrared interrupt.
24	WKEN	WAKE interrupt enable. Enables/disables the WAKE bit to generate an interrupt. The WAKE bit is set at the detection of a start bit by the receiver. 0 Disable the WAKE interrupt. 1 Enable the WAKE interrupt.
25	IDDMAEN	DMA IDLE condition detected interrupt enable. Enables/disables the receive DMA request for the IDLE interrupt (triggered with IDLE flag in SSR2[12]). 0 Disable the DMA IDLE interrupt. 1 Enable the DMA IDLE interrupt.
26	IRSC	IR special case. Selects the clock for the vote logic. When set, IRSC switches the vote logic clock from the sampling clock to the SIRI reference clock. The IR pulses are counted a predetermined amount of time depending on the reference frequency. Refer to Section 14.5.2.3, “Infrared Special Case (IRSC) Bit.” 0 The vote logic uses the sampling clock (16x baud rate) for normal operation. 1 The vote logic uses the SIRI reference clock.
27	LPBYP	Low power bypass. Allows bypassing the low power new features in the SIRI. Use during debug phase. 0 Enable low power features. 1 Disable low power features.
28	TCEN	Transmit complete interrupt enable. Enables/disables the TXDC bit to generate an interrupt. 0 Disable the TXDC interrupt. 1 Enable the TXDC interrupt.
29	BKEN	BREAK condition detected interrupt enable. Enables/disables the BRCD bit to generate an interrupt. 0 Disable the BRCD interrupt. 1 Enable the BRCD interrupt.
30	OREN	Receiver overrun interrupt enable. Enables/disables the ORE bit to generate an interrupt. 0 Disable the ORE interrupt. 1 Enable the ORE interrupt.
31	DREN	Receive data ready interrupt enable. Enables/disables the RDR bit to generate an interrupt. 0 Disable the RDR interrupt. 1 Enable the RDR interrupt.

14.4.1.7 SIRI FIFO Control Register (SFCR)

Figure 14-8 shows the SIRI FIFO control register.

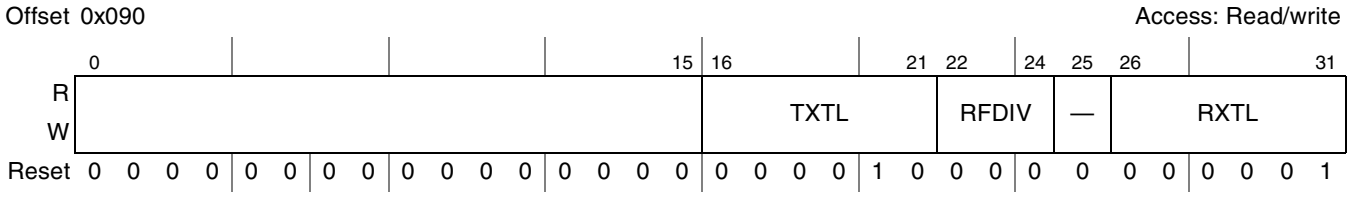


Figure 14-8. SIRI FIFO Control Register (SFCR)

Table 14-9 shows the SFCR field descriptions.

Table 14-9. SFCR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–21	TXTL	Transmitter trigger level. Controls the threshold at which a maskable interrupt or DMA request is generated by the TxFIFO. A maskable interrupt or DMA request is generated whenever the data level in the TxFIFO falls below the selected threshold. The bits are encoded as shown in the settings column. 0000_00 Reserved 0000_01 Reserved 0000_10 TxFIFO has 2 or fewer characters. ... ... 0111_11 TxFIFO has 31 or fewer characters. 1000_00 TxFIFO has 32 characters (maximum). All other settings reserved **Note:** If DMA requests are used, the value of this field must be less than the DMA transfer size. For example, with DMA transfer size of 16 bytes (16 characters), SFCR[TXTL] < 16. **Note:** If DMA requests are used, the value of this field must be less than the DMA transfer size, and it must also be less than or equal to 33 – DMA transfer size.
22–24	RFDIV	Reference frequency divider. Controls the divide ratio for the reference clock. The input clock is *siri_clkin*. The output from the divider (ref_clk) is used by BRM to create the 16x baud rate oversampling clock. 000 Divide input clock by 6. 001 Divide input clock by 5. 010 Divide input clock by 4. 011 Divide input clock by 3. 100 Divide input clock by 2. 101 Divide input clock by 1. 110 Divide input clock by 7. 111 Reserved

Table 14-9. SFCR Field Descriptions (continued)

Bits	Name	Description
25	—	Reserved, should be zero.
26–31	RXTL	Receiver trigger level. Controls the threshold at which a maskable interrupt or DMA request is generated by the RxFIFO. A maskable interrupt or DMA request is generated whenever the data level in the RxFIFO reaches the selected threshold. The RXTL bits are encoded as shown in the settings column. 00_0000 0 characters received. 00_0001 RxFIFO has 1 character. ... ... 01_1111 RxFIFO has 31 characters. 10_0000 RxFIFO has 32 characters (maximum). All other settings reserved **Note:** If DMA requests are used, the value of this field must be > 32 – DMA transfer size, and it must also be ≥ DMA transfer size. For example, with DMA transfer size of 16 bytes (16 characters), SFCR[RXTL] ≥ 17, and with DMA transfer size of 8 bytes (8 characters), SFCR[RXTL] ≥ 25.

14.4.1.8 SIRI Status Register 1 (SSR1)

Figure 14-9 shows the SIRI status register 1. Note that all non-reserved fields are write-one-to-clear.

Offset 0x094 Access: Mixed

	0															15
R W	—															
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28			31
R	PARITYERR	—	TRDY	—	ESCF	FRAMERR	RRDY	AGTIM	—	RXDS	AIRINT	AWAKE	—			
W	w1c				w1c	w1c		w1c			w1c	w1c				
Reset	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0

Figure 14-9. SIRI Status Register 1 (SSR1)

Table 14-10 shows the field descriptions of SSR1.

Table 14-10. SSR1 Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	PARITYERR	Parity error interrupt flag. Indicates a parity error is detected. Clear PARITYERR by writing 1 to it. Writing 0 to PARITYERR has no effect. When parity is disabled, PARITYERR always reads 0. At reset, PARITYERR is set to 0. 0 No parity error detected. 1 Parity error detected.
17	—	Reserved

Table 14-10. SSR1 Field Descriptions (continued)

Bits	Name	Description
18	TRDY	Transmitter ready interrupt/DMA flag. Indicates that the TxFIFO emptied below its target threshold and requires data. TRDY is automatically cleared when the data level in the TxFIFO goes above the set threshold level by TXTL bits. At reset, TRDY is set to 1. 0 The transmitter does not require data. 1 The transmitter requires data (interrupt posted).
19	—	Reserved
20	ESCF	Escape sequence interrupt flag. Indicates if an escape sequence was detected. ESCF is asserted when the ESCEN bit is set and an escape sequence is detected in the RxFIFO. Clear ESCF by writing 1 to it. Writing 0 to ESCF has no effect. 0 No escape sequence detected. 1 Escape sequence detected.
21	FRAMERR	Frame error interrupt flag. Indicates that a frame error was detected. An interrupt is generated by this. Clear FRAMERR by writing 1 to it. Writing 0 to FRAMERR has no effect. 0 No frame error detected. 1 Frame error detected.
22	RRDY	Receiver ready interrupt/DMA flag. Indicates that the RxFIFO data level is above the threshold set by the RXFL bits. (See the RXFL bits description in Table 14-9 for setting the interrupt threshold.) When asserted, RRDY generates a maskable interrupt or DMA request. RRDY is automatically cleared when data level in the RxFIFO goes below the set threshold level. At reset, RRDY is set to 0. 0 No character ready. 1 Character(s) ready (interrupt posted).
23	AGTIM	Aging timer interrupt flag. Indicates that data in the RxFIFO has been idle for a time of 8 character lengths (where a character length consists of 7 or 8 bits, depending on the setting of the WS bit in SCR2, with the bit time corresponding to the baud rate setting) and FIFO data level is less than RxFIFO threshold level (RxTL in the SFCR). Clear AGTIM by writing 1 to it. 0 AGTIM is not active 1 AGTIM is active.
24	—	Reserved
25	RXDS	Receiver IDLE interrupt flag. Indicates that the receiver state machine is in an IDLE state, the next state is IDLE, and the receive pin is high. RXDS is automatically cleared when a character is received. RXDS is active only when the receiver is enabled. 0 Receive is in progress. 1 Receiver is IDLE.
26	AIRINT	Asynchronous IR WAKE interrupt flag. Indicates that the IR WAKE pulse was detected. Clear AIRINT by writing 1 to it. Writing 0 to AIRINT has no effect. 0 No pulse was detected 1 A pulse was detected
27	AWAKE	Asynchronous WAKE interrupt flag. Indicates that a falling edge was detected. Clear AWAKE by writing 1 to it. Writing 0 to AWAKE has no effect. Caution: AWAKE Interrupt flag cannot be used in loopback mode (STS[12] = 1'b1) as RXD pin will be ignored. (See Table 14-18.) 0 No falling edge was detected on the RXD serial pin. 1 A falling edge was detected on the RXD serial pin.
28–31	—	Reserved

14.4.1.9 SIRI Status Register 2 (SSR2)

Figure 14-10 shows the SIRI status register 2.

Offset 0x098 Access: Mixed

	0															15
R	—															
W																
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ADET	TXFE	—	IDLE	ACST	—		IRINT	WAKE	—			TXDC	BRCD	ORE	RDR
W	w1c			w1c	w1c			w1c	w1c					w1c	w1c	
Reset	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	0

Figure 14-10. SIRI Status Register 2 (SSR2)

Table 14-11 shows the SSR2 field descriptions.

Table 14-11. SSR2 Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	ADET	Automatic baud rate detect complete. Indicates that an “A” or “a” was received and that the receiver detected and verified the incoming baud rate. Clear ADET by writing 1 to it. Writing 0 to ADET has no effect. 0 ASCII “A” or “a” was not received. 1 ASCII “A” or “a” was received.
17	TXFE	Transmit buffer FIFO empty. Indicates that the transmit buffer (TxFIFO) is empty. TXFE is cleared automatically when data is written to the TxFIFO. Even though TXFE is high, the transmission might still be in progress. 0 The transmit buffer (TxFIFO) is not empty. 1 The transmit buffer (TxFIFO) is empty.
18	—	Reserved
19	IDLE	Idle condition. Indicates that an idle condition has existed for more than a programmed amount frame. (See Section 14.5.2.7.1, “Idle Line Detect.”) An interrupt can be generated by this IDLE bit if IDEN (SCR1[12]) is enabled.Clear IDLE by writing 1 to it; writing 0 to IDLE has no effect. 0 No idle condition detected. 1 Idle condition detected.
20	ACST	Autobaud counter stopped. in autobaud detection (ADBR=1). Indicates the counter that determines the baud rate was running and is now stopped. This means either START bit is finished (if ADNIMP=1), or Bit 0 is finished (if ADNIMP=0). See Section , “New Autobaud Counter Stopped Bit and Interrupt” for more details. An interrupt can be flagged if ACIEN=1. Clear ACST by writing 1 to it. 0 Measurement of bit length not finished (in autobaud). 1 Measurement of bit length finished (in autobaud).
21–22	—	Reserved
23	IRINT	Serial infrared interrupt flag. When an edge is detected on the RX pin, this flag will be asserted. This flag can cause an interrupt that can be masked using the control bit SCR4[ENIRI]. Clear IRINT by writing 1 to it. 0 No edge detected. 1 Valid edge detected.

Table 14-11. SSR2 Field Descriptions (continued)

Bits	Name	Description
24	WAKE	Wake. Indicates the start bit is detected. WAKE can generate an interrupt that can be masked using the WKEN bit. Clear WAKE by writing 1 to it. Writing 0 to WAKE has no effect. 0 Start bit not detected. 1 Start bit detected.
25–27	—	Reserved
28	TXDC	Transmission complete. Indicates that the transmit buffer (TxFIFO) and shift register are empty; therefore, the transmission is complete. TXDC is cleared automatically when data is written to the TxFIFO. 0 Transmission is incomplete. 1 Transmission is complete.
29	BRCD	BREAK condition detected. Indicates that a BREAK condition was detected by the receiver. Clear BRCD by writing 1 to it. Writing 0 to BRCD has no effect. 0 No BREAK condition was detected. 1 A BREAK condition was detected.
30	ORE	Overrun error. When set to 1, ORE indicates that the receive buffer (RxFIFO) was full (32 chars inside), and a 33rd character has been fully received. This 33rd character has been discarded. Clear ORE by writing 1 to it. Writing 0 to ORE has no effect. 0 No overrun error. 1 Overrun error.
31	RDR	Receive data ready. Indicates that at least 1 character is received and written to the RxFIFO. If the SRXD register is read and there is only 1 character in the RxFIFO, RDR is automatically cleared. 0 No receive data ready. 1 Receive data ready.

14.4.1.10 SIRI Escape Character Register (SESC)

Figure 14-11 shows the SIRI escape character register.

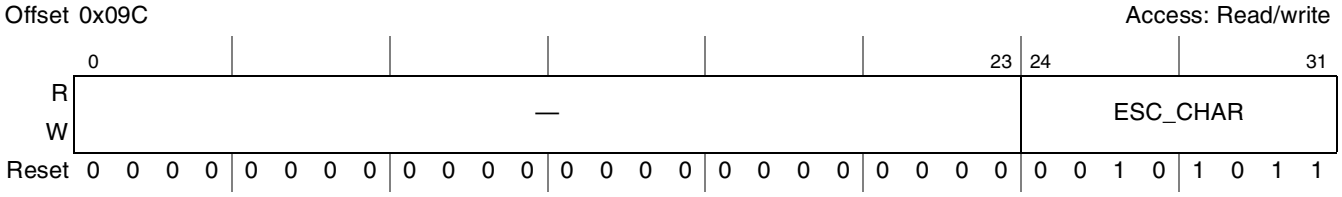


Figure 14-11. SIRI Escape Character Register (SESC)

Table 14-12 shows the SESC field descriptions.

Table 14-12. SESC Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24–31	ESC_CHAR	SIRI escape character. Holds the selected escape character that all received characters are compared against to detect an escape sequence.

14.4.1.11 SIRI Escape Timer Register (STIM)

Figure 14-12 shows the SIRI escape timer register.

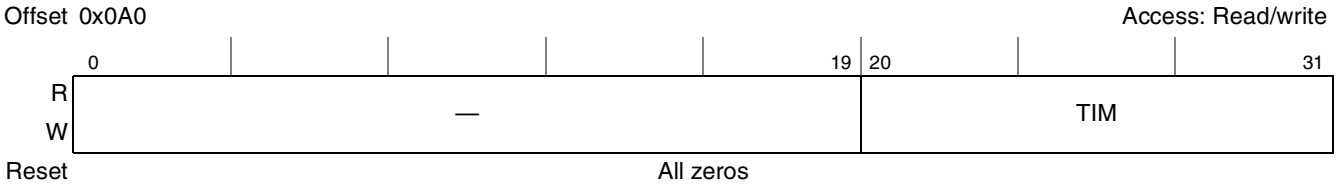


Figure 14-12. SIRI Escape Timer Register (STIM)

Table 14-13 shows the STIM field descriptions.

Table 14-13. STIM Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–31	TIM	SIRI escape timer. Holds the maximum time interval (in ms) allowed between escape characters. The escape timer register is programmable in intervals of 2 ms. See Section 14.5.2.10, “Escape Sequence Detection” and Table 14-32 for more information on the escape sequence detection. Reset value 0x000 = 2 ms up to 0xFFF = 8.192 s.

14.4.1.12 SIRI BRM Incremental Register (SBIR)

Figure 14-13 shows the SIRI BRM incremental register.

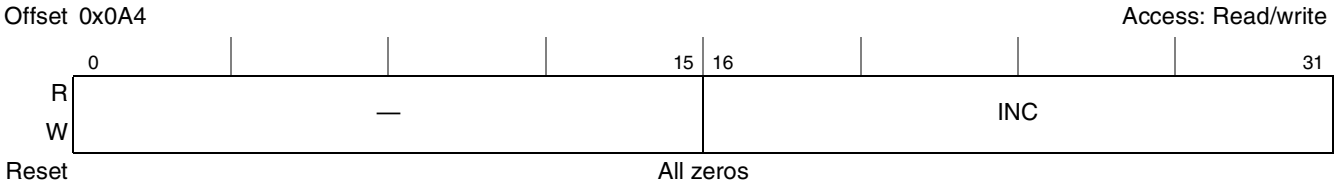


Figure 14-13. SIRI BRM Incremental Register (SBIR)

Table 14-14 shows the SBIR field descriptions.

Table 14-14. SBIR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	INC	Incremental numerator. Holds the numerator value minus one of the BRM ratio. See Section 14.5.2.8, “Binary Rate Multiplier (BRM).” The SBIR register MUST be updated before the SBMR register is updated for the baud rate to be updated correctly. If only one register is written to by the software, the BRM ignores this data until the other register is written to by the software. Updating this field using byte accesses is not recommended and is undefined.

14.4.1.13 SIRI BRM Modulator Register (SBMR)

Figure 14-14 shows the SIRI BRM modulator register.

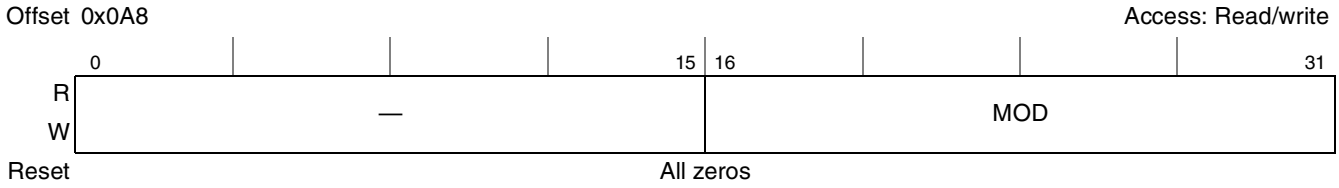


Figure 14-14. SIRI BRM Modulator Register (SBMR)

Table 14-15 shows the SBMR field descriptions.

Table 14-15. SBMR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	MOD	Modulator denominator. Holds the value of the denominator minus one of the BRM ratio. See Section 14.5.2.8, “Binary Rate Multiplier (BRM).” The SBIR register MUST be updated before the SBMR register is updated for the baud rate to be updated correctly. If only one register is written to by the software, the BRM ignores this data until the other register is written to by the software. Updating this register using byte accesses is not recommended and undefined.

14.4.1.14 SIRI Baud Rate Count Register (SBRC)

Figure 14-15 shows the SBRC register

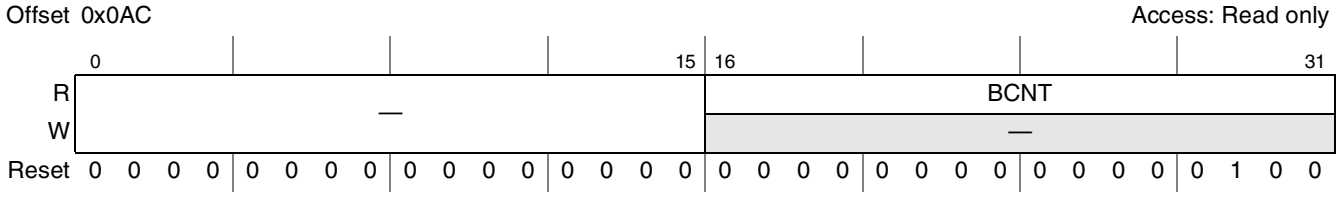


Figure 14-15. SIRI Baud Rate Count Register (SBRC)

Table 14-16 shows the SBRC field descriptions.

Table 14-16. SBRC Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	BCNT	Baud rate count register. This read-only register is used to count the start bit of the incoming baud rate (if ADNIMP=1), or start bit + bit0 (if ADNIMP=0). When the measurement is done, the baud rate count register contains the number of the SIRI internal clock cycles (clock after divider) present in an incoming bit. BCNT retains its value until the next automatic baud rate detection sequence has been initiated. The 16-bit baud rate count register is reset to 4 and stays at hex FFFF in case of an overflow.

14.4.1.15 SIRI One Millisecond Register (SONEMS)

Figure 14-16 shows the SIRI one millisecond register.

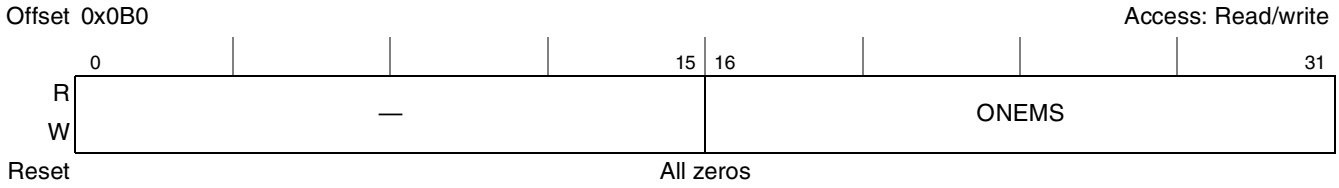


Figure 14-16. SIRI One Millisecond Register (SONEMS)

Table 14-17 shows the SONEMS field descriptions.

Table 14-17. SONEMS Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–31	ONEMS	One millisecond register. This 16-bit register must contain the value of the SIRI internal frequency (ref_clk in Figure 14-24) divided by 1000. The internal frequency is obtained after the SIRI BRM internal divider (F (ref_clk) = F(*siri_clkin*) / RFDIV). This register contains the value corresponding to the number of the SIRI BRM internal clock cycles present in one millisecond. The SONEMS (and STIM) registers value are used in the escape character detection feature (Section 14.5.2.10, “Escape Sequence Detection”) to count the number of clock cycles left between two escape characters. The SONEMS register is also used in infrared special case mode (IRSC = SCR4[5] = 1’b1). See Section 14.5.2.3, “Infrared Special Case (IRSC) Bit.”

14.4.1.16 SIRI Test Register (STS)

Figure 14-17 shows the STS register

Figure 14-17. SIRI Test Register (STS)

Table 14-18 shows the STS field descriptions.

Table 14-18. STS Field Descriptions

Bits	Name	Description
0–17	—	Reserved
18	FRCPERR	Force parity error. Forces the transmitter to generate a parity error if parity is enabled. FRCPERR is provided for system debugging. 0 Generate normal parity. 1 Generate inverted parity (error).
19	LOOP	Loop TX and RX for test. Controls loopback for test purposes. When LOOP is high, the receiver input is internally connected to the transmitter and ignores the RXD pin. The transmitter is unaffected by LOOP. 0 Normal receiver operation. 1 Internally connect the transmitter output to the receiver input.
20	—	Reserved
21	LOOPIR	Loop TX and RX for IR test (LOOPIR). This bit controls loopback from transmitter to receiver in the Infrared interface. 0 No IR loop. 1 Connect IR transmitter to IR receiver.
22–24	—	Reserved
25	TXEMPTY	TxFIFO empty. Indicates that the TxFIFO is empty. 0 The TxFIFO is not empty. 1 The TxFIFO is empty.
26	RXEMPTY	RxFIFO empty. Indicates the RxFIFO is empty. 0 The RxFIFO is not empty. 1 The RxFIFO is empty.
27	TXFULL	TxFIFO full. Indicates the TxFIFO is full. 0 The TxFIFO is not full. 1 The TxFIFO is full.
28	RXFULL	RxFIFO full. Indicates the RxFIFO is full. 0 The RxFIFO is not full. 1 The RxFIFO is full.
29–30	—	Reserved
31	SOFTRST	Software reset. Indicates the status of the software reset ($\overline{\text{SRST}}$ bit of SCR2). 0 Software reset is inactive. 1 Software reset is active.

14.4.1.17 FIRI Transmitter Control Register (FIRITCR)

The transmitter control register (FIRITCR), shown in Figure 14-18, is a 32-bit read/write register which controls transmitter operation.

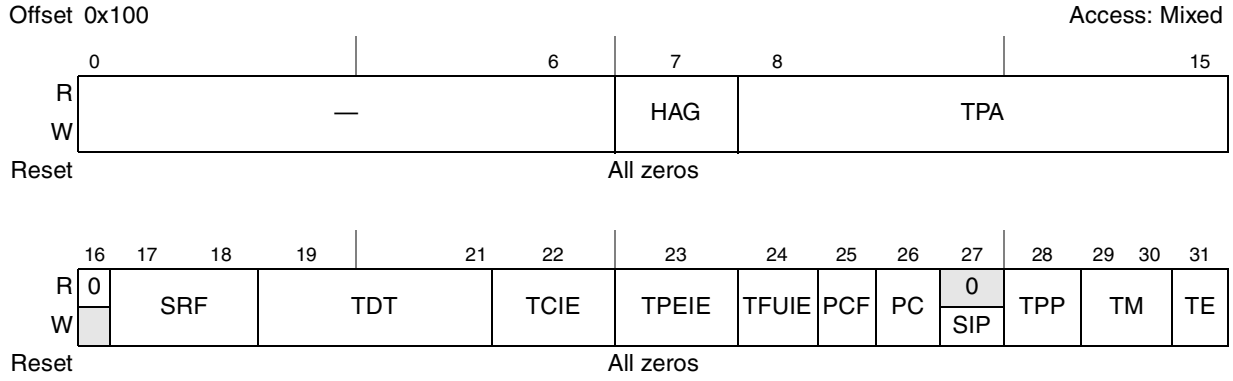


Figure 14-18. Transmitter Control Register (FIRITCR)

Table 14-19 lists the fields of FIRITCR.

Table 14-19. FIRITCR Field Descriptions

Bits	Name	Description
0–6	—	Reserved
7	HAG	Hardware address generator When this bit is set the content of TPA field is transmitted as packet address. When the bit is cleared the packet address is read from FIFO. 0 Read packet address from FIFO. 1 Use TPA field as packet address.
8–15	TPA	Transmit packet address This field contains the 8-bit transmit packet address. If HAG is cleared, the TPA field has no effect.
16	—	Reserved
17–18	SRF	Start field repeat factor This field contains the number of PA or STA fields transmitted at the beginning of the packet for FIR and MIR mode respectively. 00 16 PA or 2 STA fields are transmitted. 01 32 PA or 4 STA fields are transmitted. 10 64 PA or 8 STA fields are transmitted. 11 128 PA or 16 STA fields are transmitted.

Table 14-19. FIRITCR Field Descriptions (continued)

Bits	Name	Description
19–21	TDT	Transmitter DMA request trigger level This field controls the FIFO depth that triggers a DMA request. The burst length value (BL) should not exceed the number of vacant bits in the FIFO which is controlled by TDT. 0 DMA request generated when FIFO is empty 1 DMA request generated when FIFO contain 16 bytes of data 2 DMA request generated when FIFO contain 32 bytes of data 3 DMA request generated when FIFO contain 48 bytes of data ..... 7 DMA request generated when FIFO contain 112 bytes of data **Note:** FIRITCR[TDT] × 16 must be ≤ 128 – FIRICR[BL], and FIRITCR[TDT] × 16 must also be ≤ DMA transfer size.
22	TCIE	Transmit complete interrupt enable This bit enables TC status bits to generate a TCI interrupt. 0 TCI interrupt is not triggered by TC. 1 TCI interrupt is triggered by TC.
23	TPEIE	Transmitter packet end interrupt enable This bit enables the TPE and SIPE status bits to generate a PEI interrupt. 0 PEI interrupt is not triggered by TPE or SIPE bits. 1 PEI interrupt is triggered by TPE or SIPE bits.
24	TFUIE	Transmitter FIFO underrun interrupt enable This bit enables the TFU status bit to generate a TFU interrupt. 0 TFU interrupt disabled 1 TFU interrupt enabled
25	PCF	Packet complete by FIFO This bit determines how a packet is completed if a FIFO underrun event occurs. 0 Send CRC and STO fields. 1 Send packet abort symbol.
26	PC	Packet complete This bit determines how a packet is completed when TE bit is cleared or when SIP bit is set in the middle of the transfer. 0 Send CRC and STO fields. 1 Send packet abort symbol.
27	SIP	Transmit enable of SIP Setting this bit produces a serial infrared interaction pulse transmission. Clearing this bit is ignored. This bit is always read as zero. If this bit is set while in the middle of the transfer, the packet will be completed according to setting of the PC bit.
28	TPP	Transmitter pulse polarity bit 0 Transmitted pulse is not inverted. 1 Transmitted pulse is inverted.

Table 14-19. FIRITCR Field Descriptions (continued)

Bits	Name	Description
29–30	TM	Transmitter mode This bit controls the transmission mode. 00 FIR mode 01 0.576-Mbps MIR mode 10 1.152-Mbps MIR Mode 11 Reserved
31	TE	Transmitter enable This bit controls the FIRI transmitter. If this bit is cleared in the middle of the transfer the packet will be completed according to the setting of the PC bit. When the packet is completed, the transmitter clocks are then gated OFF. 0 Transmitter disabled 1 Transmitter enabled

14.4.1.18 FIRI Transmitter Count Register (FIRITCTR)

The transmitter count register (FIRITCTR), shown in Figure 14-19, is a 32-bit read-write register that controls packed size.

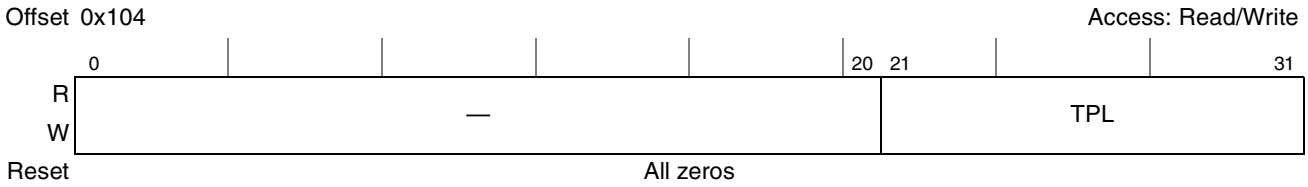


Figure 14-19. Transmitter Count Register (FIRITCTR)

Table 14-24 lists the fields of FIRITCTR.

Table 14-20. FIRITCTR Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21–31	TPL	Transmit packet length bits Length of DD field 0x0 Send 1 byte 0x1 Send 2 bytes 0x2 Send 3 bytes .... 0x7FFSend 2048 bytes

14.4.1.19 FIRI Receiver Control Register (FIRIRCR)

The receiver control register, shown in Figure 14-20, is 32-bit read-write register which controls receiver operation.

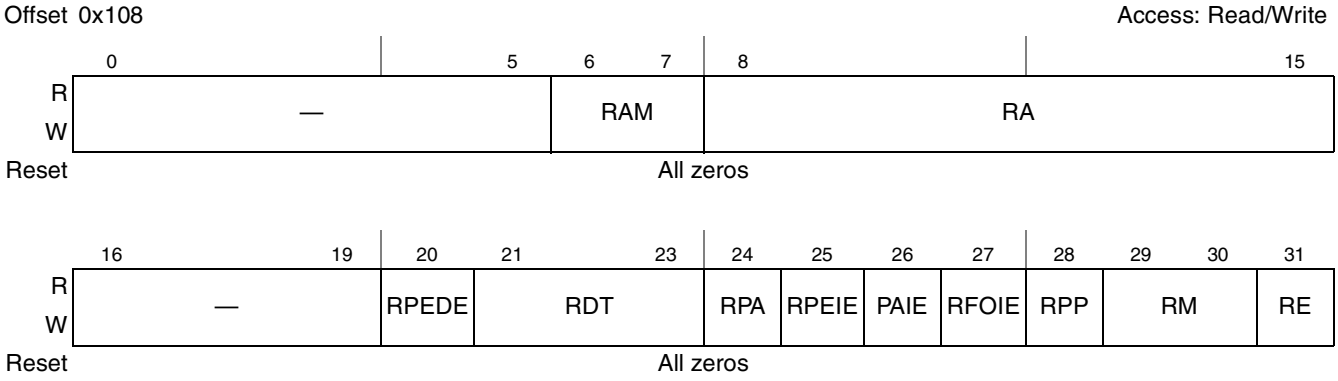


Figure 14-20. Receiver Control Register (FIRIRCR)

Table 14-21 lists the fields of FIRIRCR.

Table 14-21. FIRIRCR Field Descriptions

Bits	Name	Description
0–5	—	Reserved
6–7	RAM	Address match bit The value of this field can be changed when RE bit is cleared. 00 Don't match address. 01 Match packet address to RA field. 10 Match packet address to broadcast address. 11 Match packet address to RA field and to broadcast address.
8–15	RA	Receiver address Determines receiver packet address. If RAM field value is 00 or 10, the RA field has no effect. The value of this field can be changed when RE bit is cleared.
16–19	—	Reserved
20	RPEDE	Receiver packet end DMA request enable Enable DMA request generation at end of packet. If this bit is set the value of BL should not be grater than DD + CRC size. 0 DMA request is not affected by end of packet. 1 DMA request is generated at end of packet.

Table 14-21. FIRIRCR Field Descriptions (continued)

Bits	Name	Description
21–23	RDT	Receiver DMA request trigger level Sets minimum FIFO depth that will trigger DMA request. The value of BL should not exceed the trigger level selected by RDT. 0 Reserved 1 DMA request generated when FIFO contains 16 bytes of data 2 DMA request generated when FIFO contains 32 bytes of data 3 DMA request generated when FIFO contains 48 bytes of data ..... 7 DMA request generated when FIFO contains 112 bytes of data **Note:** FIRIRCR[RDT] × 16 must be ≥ 128 – FIRICR[BL], and FIRIRCR[RDT] × 16 must also be ≥ DMA transfer size. FIRIRCR[RDT]*16 must also be ≥ 128 – DMA transfer size.
24	RPA	Receiver packet abort Determines behavior of FIFO upon detection of an illegal symbol When an illegal symbol is detected, the DDE or CRCE field is set, and if RPA is set, the receive FIFO pointers are cleared and the receiver starts to search for PA or STA fields for FIR and MIR mode respectively. If RPA is cleared, the receiver continues to write to the FIFO. 0 Do not clear the FIFO upon detection of illegal symbol. 1 Clear the FIFO upon detection of illegal symbol.
25	RPEIE	Receiver packet end interrupt enable Enable the RPE status bits to cause PEI interrupt. 0 PEI interrupt is not triggered by RPE bit. 1 PEI interrupt is triggered by RPE bit.
26	PAIE	Packet abort interrupt enable Enable the DDE and CRCE status bits to cause PAI interrupt. 0 PAI interrupt disabled 1 PAI interrupt enabled
27	RFOIE	Receiver FIFO overrun interrupt enable Enable the RFO status bit to cause RFOI interrupt. 0 RFOI interrupt disabled 1 RFOI interrupt enabled
28	RPP	Receiver pulse polarity 0 Receiver signal is not inverted. 1 Receiver signal is inverted.
29–30	RM	Receiver mode This field controls the transmission mode. 00 FIR mode 01 0.576-Mbps MIR mode 10 1.152-Mbps MIR Mode 11 Reserved
31	RE	Receiver enable bit When this field is cleared the receiver clocks are gated OFF. 0 Receiver disabled 1 Receiver enabled

14.4.1.20 FIRI Transmit Status Register (FIRITSR)

The transmit status register is a 32-bit register which reflects the status of transmitter and FIFO. It contains read-only and write-one-to-clear bits.

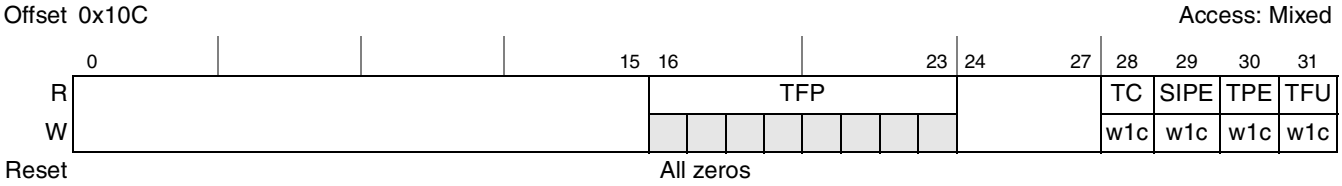


Figure 14-21. Transmit Status Register (FIRITSR)

Table 14-22 lists the fields of FIRITSR.

Table 14-22. FIRITSR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–23	TFP	Transmitter FIFO pointer Points to the bottom of transmitter FIFO. Read-only field.
24–27	—	Reserved
28	TC	Transmit complete Indicates that the transmission is completed (including CRC and STO field) and the transmitter FIFO is empty. This field is cleared by writing a one to it. 0 Transmitting is not completed 1 Transmitting is completed
29	SIPE	SIP end Indicates that a serial infrared interaction pulse has been transmitted. This field is cleared by writing a one to it. 0 SIP transmission isn't completed 1 SIP had been transmitted
30	TPE	Transmitter packet end Indicates that TPS bytes of data (DD field) have been transmitted. This field is cleared by writing a one to it. 0 DD transmissions isn't completed 1 DD had been transmitted
31	TFU	Transmitter FIFO underrun Indicates that the transmitter attempted to read from the FIFO when the FIFO was empty, and there is no valid data to transmit. This field is cleared by writing a one to it. 0 No transmitter FIFO underrun 1 Transmitter FIFO is underrun

14.4.1.21 FIRI Receive Status Register (FIRIRSR)

The receive status register is 32-bit register which reflects the status of receiver and FIFO. It contains read-only and write-one-to-clear bits:

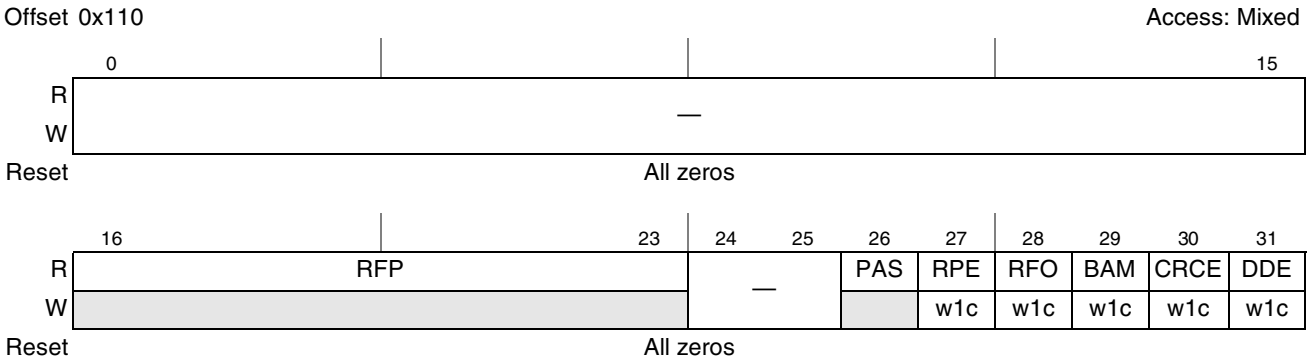


Figure 14-22. Receive Status Register (FIRIRSR)

Table 14-23 lists the fields of FIRIRSR.

Table 14-23. FIRIRSR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–23	RFP	Receiver FIFO pointer Points to bottom of receiver FIFO. Read-only field.
24–25	—	Reserved
26	PAS	Preamble search Indicates that the receiver is searching for PA + STA or STA fields for FIR and MIR modes respectively. Read-only field. 0 Receiver don't searching for PA or STA field 1 Receiver searching for PA or STA field
27	RPE	Receiver packet end Indicate that STO field or packet abort symbol (b'0000,000 and b'0000,0000 for MIR and FIR restrictively) is detected. This bit is cleared by writing a one to it. 0 STO was not detected 1 STO field is detected
28	RFO	Receiver FIFO overrun Indicates that the receiver attempted to write to the FIFO when the FIFO was full. This bit is cleared by writing a one to it. 0 Receiver FIFO is not overrun 1 Receiver FIFO is overrun
29	BAM	Broadcast address match Indicate that the address field of the packet match Broadcast Address 0xFF. This bit is cleared by writing a one to it. 0 No broadcast 1 Broadcast packet

Table 14-23. FIRIRSR Field Descriptions (continued)

Bits	Name	Description
30	CRCE	CRC error Indicate that CRC check failed. This bit is cleared by writing a one to it. 0 No CRC check failure 1 CRC check failure
31	DDE	DD error Indicates that an illegal symbol has been detected in DD or CRC32/CRC fields. This bit is cleared by writing a one to it. 0 No illegal symbols in DD or CRC field 1 Illegal symbol in DD or CRC field

14.4.1.22 FIRI Control Register (FIRICR)

The FIRI control register (FIRICR) is shown in Figure 14-23.

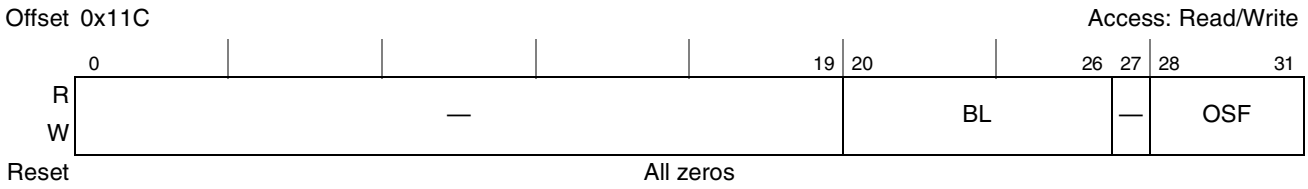


Figure 14-23. FIRI Control Register (FIRICR)

Table 14-24 lists the fields of FIRICR.

Table 14-24. FIRICR Field Descriptions

Bits	Name	Description
0–19	—	Reserved
20–26	BL	Burst length Contain number of bytes that are transferred in a DMA burst. Used to prevent FIFO overrun/underrun. DMA request released according to this field. See also description of TDT and RDT fields. 0 128 bytes 1 1 byte 2 2 bytes .... 1111_111127 bytes **Note:** FIRICR[BL] must be ≥ DMA transfer size (for both Rx and Tx DMA transfers if DMA is used for both Rx and Tx).
27	—	Reserved

Table 14-24. FIRICR Field Descriptions (continued)

Bits	Name	Description
28–31	OSF	Over sampling factor This field controls the oversampling factor of the “chip” if RE bit is set and the prescaling factor of IR_CLKIN if TE is set. Note that “chip” rate is 4 and 2 times grater then bit rate for MIR and FIR respectively. MIR Mode 0000–0100Reserved 0101 Oversample by 6 (Recommended) 0110 Oversample by 7 .... 1111 Oversample by 16 FIR Mode 0000–0100Reserved 0101 Oversample by 6 (Recommended) .... 1000 Oversample by 9 1001–1111Reserved Note that a value of 0101 (oversample by 6) is recommended.

14.5 Functional Description

14.5.1 Overview

14.5.1.1 SIRI Overview

SIRI communications are based on standard serial communications, in which each byte is sent or received one bit at a time. (User-selectable parity bits or stop bits are sent along with each byte.) Only the modulation scheme is different.

14.5.1.2 FIRI Overview

Medium and fast infrared standards use more sophisticated signaling schemes than serial infrared; they also use a packet format. The FIRI supports MIR (0.576 and 1.152 Mbit/s) and FIR (4 Mbit/s) physical layer protocols. This section provides a brief overview of the MIR and FIR standards. See the Infrared Data Association serial infrared physical layer specification, version 1.4. for more details.

14.5.1.2.1 MIR Packet Structure

The data of the MIR physical layer is organized into packets as follows:

Table 14-25. MIR Packet structure

STA	STA	DD (address, control, and data)	CRC	STO

where STA (start flag) and STO (stop flag) are identical predefined sequences (left symbol transmitted/received first):

- STA, STO—0b0111_1110
- DD—Can be up to 2048 bytes long; address (8 bits) and control are optional
- CRC—Generated according to the CRC-CCITT algorithm, defined as follows:

$$CRC(x) = x^{16} + x^{12} + x^5 + 1 \qquad \textit{Eqn. 14-1}$$

The CRC(x) value is inverted prior to transmission.

The DD and CRC fields are not transmitted as is; they are first converted according to the MIR standard, as follows:

In MIR mode, the time to transfer a bit is divided up into four portions, called ‘chips.’ A data one is transmitted as four chips of no LED pulse; a data zero is transmitted as one chip of LED pulse followed by three chips of no pulse. This substitution scheme is shown in Table 14-26, where a one in a data symbol represents an LED pulse.

Table 14-26. MIR Modulation

Data Bit	Data Symbol Sent (DD)
0	1000
1	0000

Tolerance of bit rate according to the standard must not exceed ±0.1%.

14.5.1.2.2 FIR Packet structure

The data of the FIR physical layer is organized into packets as follows:

Table 14-27. FIR Packet structure

PA	STA	DD (address, control, and data)	CRC32	STO

where PA (preamble), STA, and STO are predefined sequences (left symbol transmitted/received first):

- PA—0b1000_0000_1010_1000 repeated 16 times
- STA—0b0000_1100_0000_1100_0110_0000_0110_0000
- STO—0b0000_1100_0000_1100_0000_0110_0000_0110
- DD—Can be up to 2048 bytes long; address (8 bits) and control are optional
- CRC32—32-bit CRC generated according to the IEEE 802 CRC32 algorithm (see IEEE standard 802™), as follows:

$$CRC32(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1 \quad \textit{Eqn. 14-2}$$

  The CRC(x) value is inverted prior transmission.

The DD and CRC32 fields are not transmitted as is; they are first converted with the 4 pulse position modulation (4PPM), as follows:

In FIR mode, the time taken to transfer two data bits is divided up into four chips. Then, the four possible two-bit pairs are substituted using the scheme shown in Table 14-28, where a one in a data symbol represents an LED pulse. (Thus the name ‘four position pulse modulation’: each symbol has exactly one LED pulse, and differs from the other symbols only in the location of that pulse.)

Table 14-28. 4PPM Mapping

Data Bit Pair	4PPM Data Symbol (DD)
00	1000
01	0100
10	0010
11	0001

Tolerance of chip rate according to the standard should not exceed ±0.01%. Pulse width tolerance of the electrical signal driven by the IR detector (photo diode) can be greater than the tolerance of the optical signal defined by IrDA and can depend on the LED and IR devices used with the FIRI.

14.5.2 SIRI Operation

The sections gives a general overview of SIRI communications using this controller. Figure 14-24 is a block diagram of the SIRI module.

Note that the input clock, referred to as *siri_clkin* in this document, is the platform clock (MPX clock) divided by two.

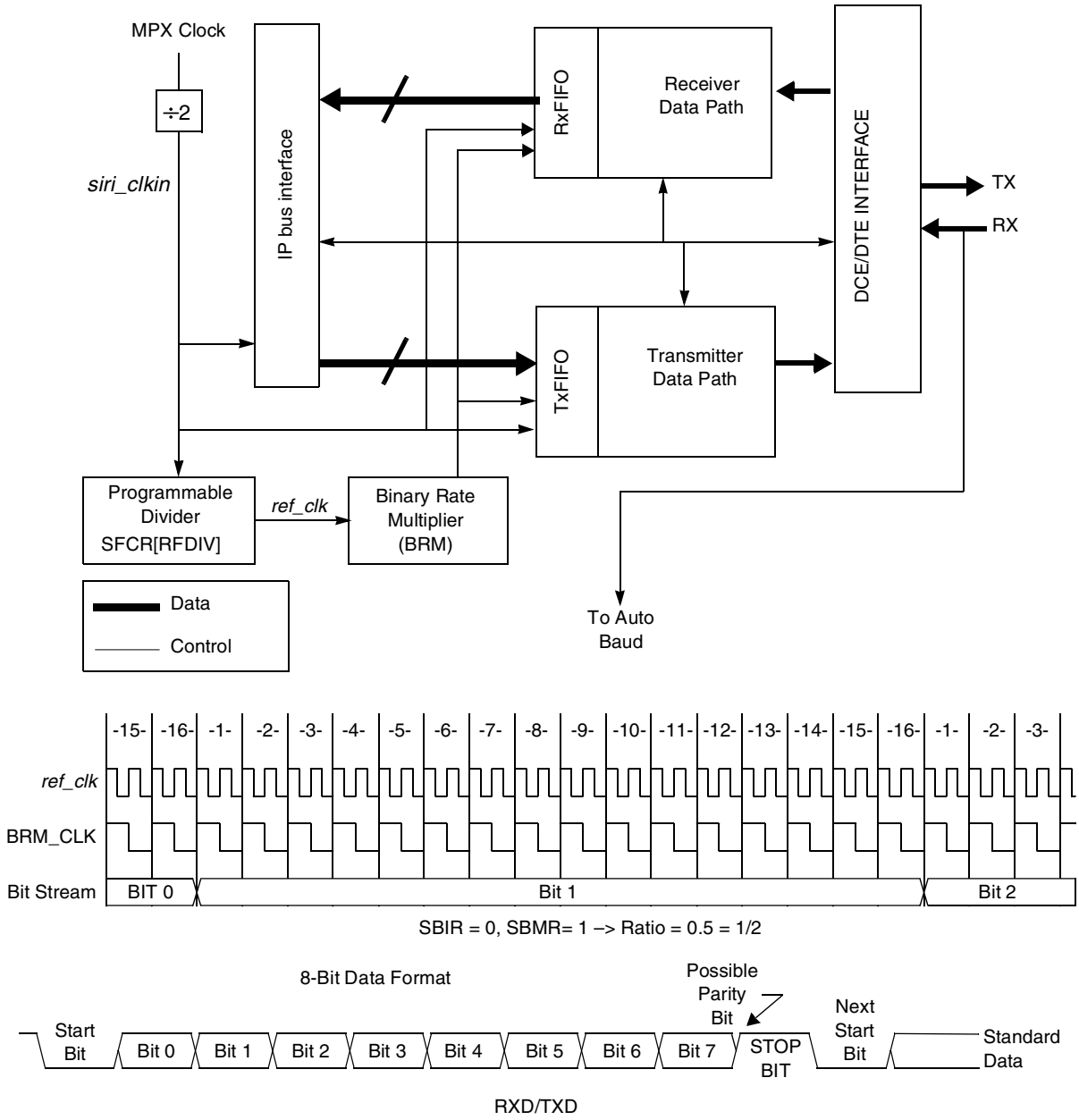


Figure 14-24. SIRI Block Diagram

14.5.2.1 Generalities

The serial infrared interface is selected when SCR1[IREN] is set. It is compatible with the IrDA serial infrared physical layer specification, in which a zero is represented by a positive pulse, and a one is represented by no pulse (line remains low).

The SIRI modules operate as follows:

- Transmit—For each zero to be transmitted, a narrow positive pulse that is 3/16 of a bit time is generated. For each one to be transmitted, no pulse is generated (output is low). External circuitry must be provided to drive an infrared LED.
- Receive—When receiving, a narrow negative pulse is expected for each zero transmitted while no pulse is expected for each one transmitted. (Input is high.)

NOTE

The SIRI receiver expects to receive an inverted signal compared to the IrDA specification. Circuitry external to the SIRI transforms the infrared signal to an electrical signal.

The behavior of the SIRI is determined by three bits—SCR3[INVT], SCR4[INVR], and SCR4[IRSC], described in the following sections.

14.5.2.2 Inverted Transmission and Reception Bits (INVT and INVR)

The values of INVT and INVR depend on the IrDA transceiver connected to the SIRI. If this transceiver is not inverting on both paths Tx and Rx, a zero is represented by a positive pulse and a one is represented by no pulse. (The line remains low.) In this case, the bit INVT must be cleared and the bit INVR must be set (because Rx IR block expects an inverted signal).

The user must set INVT=1 and INVR= 0 if both paths of the transceiver are inverting, that is, a zero is represented as a negative pulse and a one is represented by no pulse (line remains high). The transceiver can also be inverting on only one path (Tx or Rx). In this case, INVT and INVR must be together equal to one or zero, depending on which path is inverted.

14.5.2.3 Infrared Special Case (IRSC) Bit

The setting of SCR4[IRSC] is based on two parameters: the baud rate and the minimum pulse duration (MPD) of the transceiver. According to the IrDA specification, for serial IR baud rates from 2.4 Kbit/s to 115.2 Kbit/s, this nominal pulse duration is equal to 3/16 of a bit duration (at the selected baud rate). For all baud rates, a minimum pulse duration is also specified. According to the IrDA specification, a zero is represented by a light pulse, so the IrDA transceiver cannot emit a light pulse shorter than the MPD. For SIR, the MPD is constant and equal to 1.41 µs.

The user must take into account the electrical MPD associated with the transceiver on the receiver path. Typically, this value is 2.0 µs, but for some manufacturers, MPD can be as low as 1.0 µs.

A zero is not only detected by the state of the RXD line, but also by the duration of the pulse. This pulse duration can be measured with 2 different clocks. In this case, the clock is selected with the IRSC bit.

- If IRSC = 0, the clock used is the BRM clock.

- If IRSC = 1, the clock used is the SIRI internal clock (*siri_clkin* after the divider (RFDIV)).

In normal operation, IRSC = 0. This means at any time, the user must ensure the frequency of BRM_clock is high enough to measure the pulse. In the SIRI and for IRSC= 0, the pulse must last at least 2 BRM clock cycles.

If this condition is not fulfilled, IRSC must be set.

The following are two examples with the minimum pulse duration equal to the MPD of the IrDA specification (in SIR).

Example 14-1. Calculation of BRM Clock Period (Clock Period < 1.41 µs)

In this instance, the user wants to receive IrDA data at 115.2 Kbit/s. The SBIR and SBMR registers are set to create the BRM_clock with a frequency of 16 × baud rate = 16 × 115.2 = 1.843 MHz. At the same time, to correctly detect the pulse, the user must ensure that 2 × BRM_clock period is lower than 1.41 us. Confirm the calculation as follows:

BRM_clock period = 1/1843000 = 542 ns

Thus, 2 × BRM_clock period = 1.09 µs < 1.41 µs, which is a good result.

Example 14-2. Calculation of BRM Clock Period (Clock Period > 1.41 µs)

In this instance, the user wants to receive IrDA data at 19.2 Kbit/s. The BRM_clock is set to 16 × 19200 = 307.2 kHz. Confirm if 2 × BRM_clock period < 1.41 µs:

BRM_clock period =1/307200 = 3.25 µs

Thus, 2 × BRM_clock period = 6.50 µs > 1.41 µs, which is not a good result.

So, in the last case, the BRM clock cannot be used to measure the pulse duration and the user must select the SIRI internal clock by setting IRSC.

NOTE

For escape character detection, when the IR special case is enabled (IRSC = 1), the SIRI must measure a duration. To do that, the user must fill the SONEMS register. See Section 14.5.2.10, “Escape Sequence Detection.”

14.5.2.4 IrDA Interrupt

The SIRI uses an edge-triggered interrupt flag (SSR2[IRINT]). When INVR = 0, detection of a falling edge on the RXD pin asserts the IRINT bit. When INVR = 1, detection of a rising edge on the RXD pin asserts the IRINT bit. When IRINT and ENIRI bits are both asserted, an interrupt is asserted. Clear the IRINT bit by setting it; writing zero to the IRINT bit has no effect.

14.5.2.5 IrDA Conclusion

Before using the SIRI, the baud rate limit must be calculated. This baud rate limit determines if IRSC must be set or not.

As already described, if IRSC = 0, the following condition must always be fulfilled:

Calculation of Baud Rate ***Eqn. 14-3***

$$2 \times \text{BRMClockPeriod} < \text{MinPulseDuration}$$

So,

$$\text{BRMClockFrequency} > \frac{2}{\text{MPD}}$$

So, knowing BRM_clock frequency = 16 × baud rate, the following:

$$\text{BaudRate} > \frac{1}{8 \times \text{MinPulseDuration}}$$

The user needs to clear IRSC under the following conditions:

- If minimum pulse duration = 2.5 µs and baud rate > 50 Kbit/s.
- If minimum pulse duration = 2.0 µs and baud rate > 62.5 Kbit/s.
- If minimum pulse duration = 1.41 µs and baud rate > 88.6 Kbit/s.

NOTE

For baud rates lower than the limit, IRSC must be set.

14.5.2.6 SIRI Transmitter

The transmitter accepts a parallel character from the core and transmits it serially along with any start, stop, and parity bits. The transmitter sends a character as soon as it is ready to transmit. Generation of BREAK characters and parity errors (for debugging purposes) is supported. The transmitter operates from the clock provided by the BRM.

The transmitter FIFO (TxFIFO) contains 32 bytes. Data is written to TxFIFO by writing to STXD[TX_DATA]. The data is written consecutively if the TxFIFO is not full. It is read (internally) consecutively if the TxFIFO is not empty. STS[TXFULL] can be used to determine whether TxFIFO is full or not.

14.5.2.6.1 Transmitter FIFO Empty Interrupt Suppression

The transmitter FIFO empty interrupt suppression logic suppresses the TXFE interrupt between writes to the TxFIFO. When TxFIFO is empty, the software can either send one or several characters. If software sends one character, it would write the character into the STXD register; then that character is immediately transferred to the transmitter shift register, assuming the transmitter is already enabled. Without interrupt suppression logic, the TXFE interrupt would be set immediately. But with this logic, for example, the interrupt is set when the last bit of the character has been transmitted before the transmission of the parity bit (if parity is enabled) and any stop bit(s).

The suppression logic does not immediately send the TXFE interrupt. It allows the software to write another character to the TxFIFO before the interrupt is asserted.

When the transmitter shift register empties before another character is written to the TxFIFO, the interrupt is asserted. Writing data to the TxFIFO would release the interrupt. The interrupt is asserted with the following conditions:

- System reset
- SIRI software reset
- When a single character has been written to transmitter FIFO and then the transmitter FIFO and the transmitter shift register become empty until another character is written to the transmitter FIFO
- That the last character in the TxFIFO is transferred to the shift register, when TxFIFO contains two or more characters. See Figure 14-25.

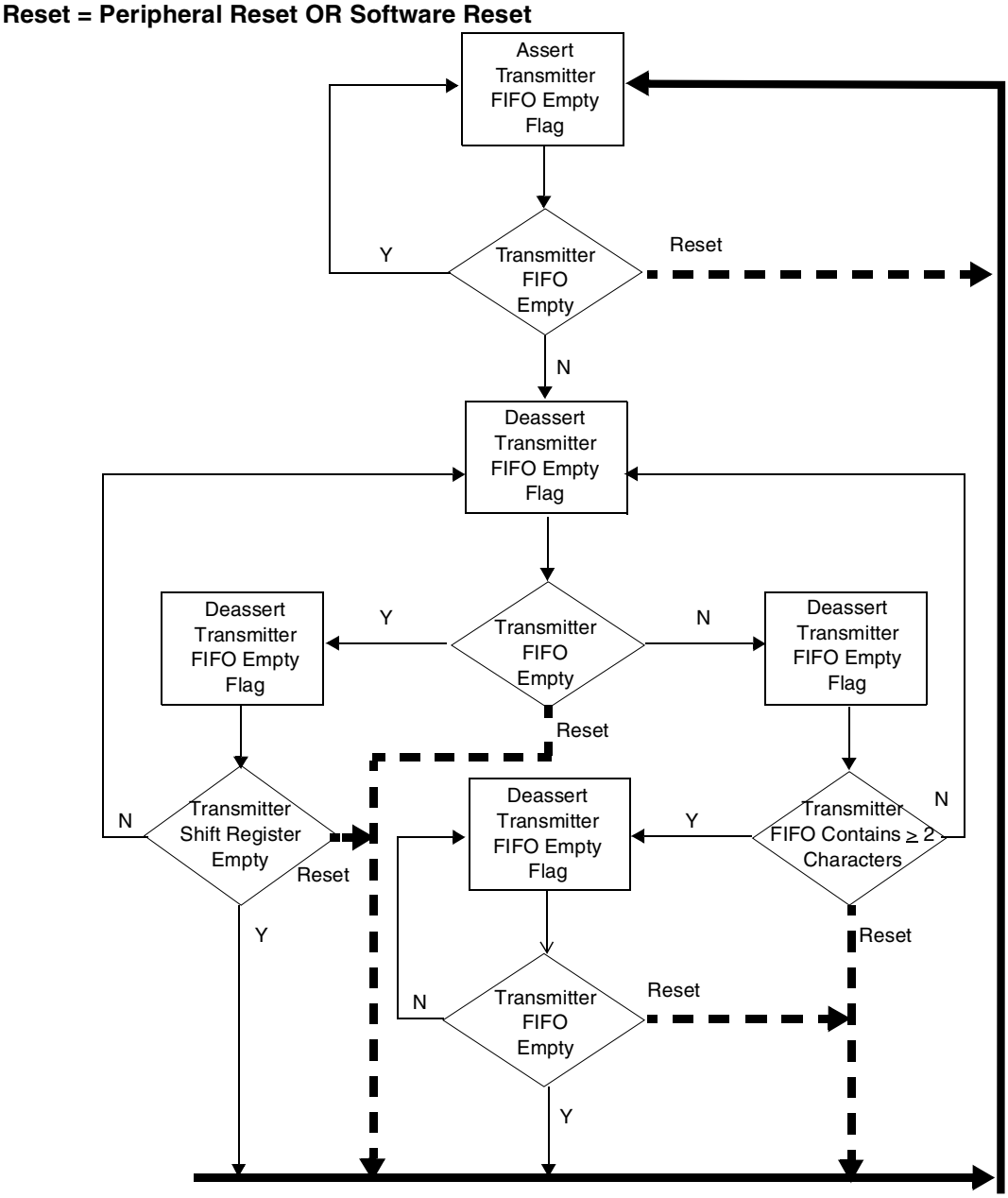


Figure 14-25. Transmitter FIFO Empty Interrupt Suppression Flowchart

14.5.2.6.2 Transmitting a Break Condition

Setting SCR1[SNDBRK] forces the transmitter to send a break character (continuous zeros). The transmitter finishes sending the character in progress (if any) before sending BREAK until this bit is reset. The user is responsible for ensuring that this bit is high long enough for it to generate a valid BREAK. The transmitter samples SNDBRK after every bit is transmitted. Following completion of the BREAK transmission, the SIRI transmits two mark bits. The user can continue to fill the FIFO. Any character remaining is transmitted when the break is terminated.

14.5.2.7 SIRI Receiver

See Figure 14-26 for the receiver flowchart. The receiver accepts a serial data stream and converts it into parallel characters. When enabled, it searches for a start bit, qualifies it, and samples the following data bits at the bit-center. Jitter tolerance and noise immunity are provided by sampling at a 16x rate and using voting techniques to clean up the samples. Once the start bit is found, the data bits, parity bit (if enabled), and stop bits (either 1 or 2 depending on user selection) are shifted in. Parity is checked and its status reported in the SRXD register when parity is enabled. Frame errors and BREAKs are also checked and reported.

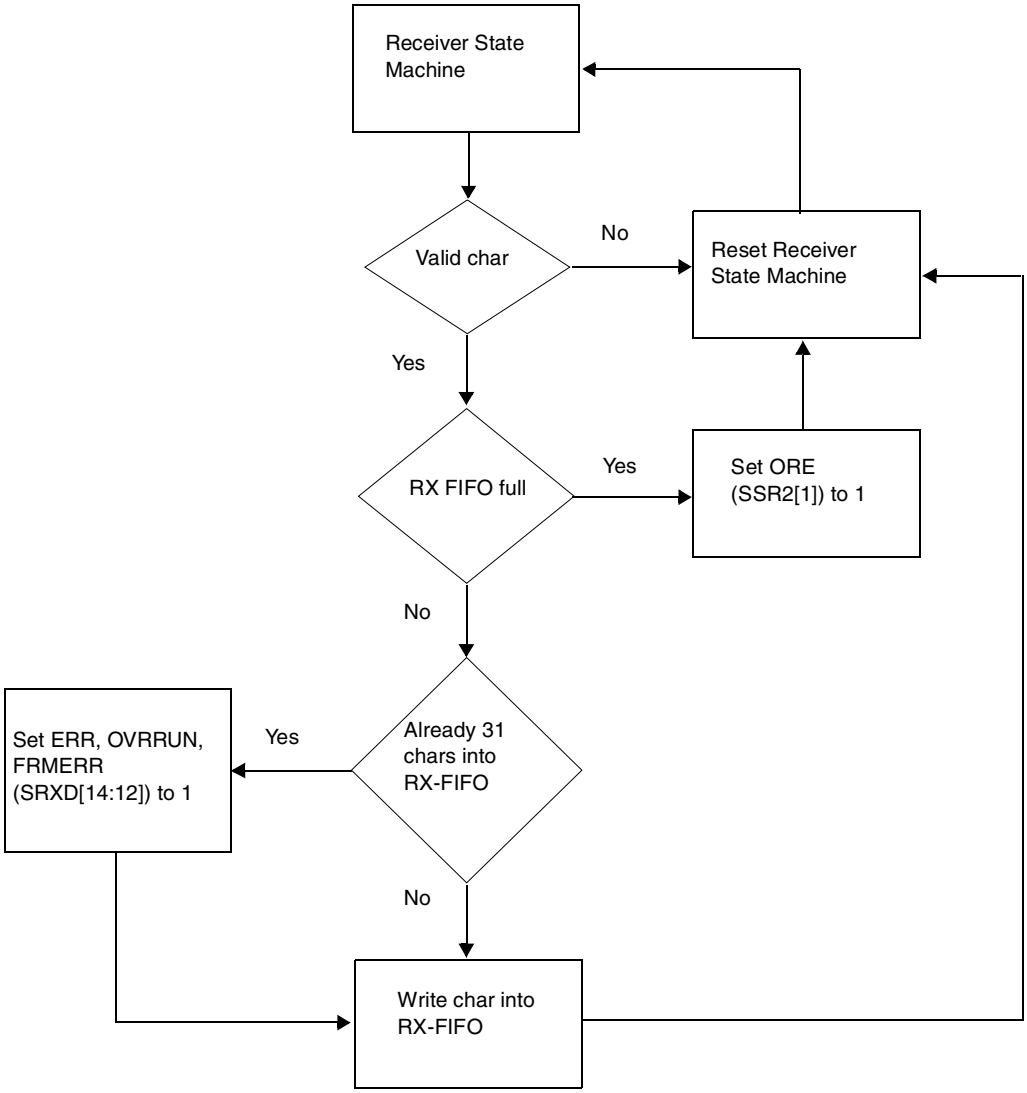


Figure 14-26. Receiver Flowchart

When a new character is ready to be read by the core from the RxFIFO, SSR2[RDR] (receive data ready) is set; an interrupt is posted if SCR4[DREN] is set. If SFCR[RXTL] = 2 (the receiver trigger level is set to 2) and two characters have been received into RxFIFO, the receiver ready interrupt flag SSR1[RRDY] is set, and an interrupt is posted if SCR1[RRDYEN] is set. If the SIRI receiver register (SRXD) is read

once, there is only one character in the RxFIFO, the interrupt generated by the RRDY bit is automatically cleared. The RRDY bit is cleared when the data in the RxFIFO falls below the programmed trigger level.

The RxFIFO contains 32 half-word entries. Characters received are written consecutively into this FIFO. If the FIFO is full and a 33rd character is received, this character is ignored and the overflow field (SSR2[ORE]) is set.

14.5.2.7.1 Idle Line Detect

The receiver logic block includes the ability to detect an idle line. Idle lines indicate the end or the beginning of a message.

For an idle condition to occur, both of the following conditions apply:

- RxFIFO must be empty.
- RXD must be idle for more than a configured number of frames (SCR1[ICD]).

When the idle condition detected interrupt enable (SCR1[IDEN]) is set and the line is idle for 4 (default), 8, 16, or 32 (maximum) frames, the detection of an idle condition flags an interrupt. When an idle condition is detected, the IDLE (SSR2[12]) bit is set. Clear the IDLE bit by writing 1 to it. Writing 0 to the IDLE bit has no effect.

14.5.2.7.2 Idle Condition Detect Configuration

The idle condition detect field is SCR1[ICD]. If this field is set to 0b00, RXD must be idle for more than 4 frames before the IDLE bit is asserted. If the bits are set to 01b, RXD must be idle for more than 8 frames before the IDLE bit is asserted. If the bits are set to 10b, RXD must be idle for more than 16 frames before the IDLE bit is asserted. If the bits are set to 11b, RXD must be idle for more than 32 frames before the IDLE bit is asserted (see Table 14-29).

Table 14-29. Detection Truth Table

SCR1[IDEN]	SCR1[ICD]	IDLE	Receive Interrupt
0	XX	0	1
1	00	asserted after 4 idle frames	asserted after 4 idle frames
1	01	asserted after 8 idle frames	asserted after 8 idle frames
1	10	asserted after 16 idle frames	asserted after 16 idle frames
1	11	asserted after 32 idle frames	asserted after 32 idle frames
Note: This table assumes that no other interrupt is set at the same time this interrupt is set for the ipi_uart_rx signal. This table shows how this interrupt affects the ipi_uart_rx signal.			

During a normal message, there is no idle time between frames. When all of the information bits in a frame are logic ones, the start bit ensures that at least one logic 0 bit time occurs for each frame so that the IDLE bit is not asserted.

14.5.2.7.3 Aging Character Detect

The receiver block can also detect when at least one character has been sitting in the RxFIFO for a time corresponding to 8 characters. This aging character capability allows the SIRI to inform the core that there are fewer characters in the RxFIFO than the Rx trigger and no new character has been detected on the RXD line. The aging capability is a timer that starts to count as soon as there is one character in RxFIFO. This counter is reset when either a RxFIFO read is performed or another character has been received in RxFIFO. If neither of those two events occurs, SSR1[AGTIM] is set when the counter has measured a time corresponding to 8 characters. AGTIM is cleared by writing a 1 to it. AGTIM can flag an interrupt to the core if SCR2[ATEN] has been set.

To summarize, AGTIM is set under the following conditions:

- There is at least one character in RxFIFO.
- No read has occurred on RxFIFO and the RXD line has stayed high for a time corresponding to 8 characters.
- The RxFIFO trigger is not reached (RRDY=0).

14.5.2.7.4 Receiver Wake

SSR2[WAKE] is set when the receiver detects a qualified start bit. For this, two conditions must be fulfilled. First, a falling edge on RXD must be detected. Second, RXD must stay at low level for more than a half-bit duration. When the wake interrupt enable bit (SCR4[WKEN]) is enabled, the receiver flags an interrupt if the WAKE status bit is set. The WAKE bit is cleared by writing a one to it; writing a zero has no effect.

The recommended procedure for programming the asynchronous interrupts is performing the following:

1. Clear the interrupts by writing 1 to the appropriate bit in the SIRI status register 1 (SSR1).
2. Poll or enable the interrupt for the receiver IDLE interrupt flag (RXDS) in the SSR1. When asserted, the RXDS bit indicates to the software that the receiver state machine is in the idle state, the next state is idle, and the RXD pin is idle (high)

14.5.2.7.5 Receiving a BREAK Condition

A BREAK condition is received when the receiver detects all zeros (including a zero during the bit time of the stop bit) in a frame. The BREAK condition sets SSR2[BRCD] and writes only the first BREAK character to the RxFIFO. Clear BRCD by writing one to it; writing zero to BRCD has no effect.

Asserting BRCD can generate an interrupt. The interrupt generation can be masked using the control bit SCR4[BKEN]. Receiving a break condition also affects the following bits in the receiver register SRXD:

- SRXD[BRK]—While high, this bit indicates that the current char was detected as a break.
- SRXD[FRMERR]—The frame error bit is always be set when BRK is set.
- SRXD[PRERR]—If odd parity was selected, the parity error bit is also be set when BRK is set.
- SRXD[ERR]—The error detect bit indicates that the character present in the receive data field has an error status. This can be asserted by a break.

14.5.2.7.6 Vote Logic

The vote logic block provides jitter tolerance and noise immunity by sampling with respect to a 16x clock (BRM_CLK) and using voting techniques to clean up the samples. Voting is implemented by sampling the incoming signal constantly on the rising edge of the BRM clock; see Figure 14-27. The receiver is provided with the majority vote value, which is 2 out of the 3 samples. Table 14-30 shows examples of majority vote results of the vote logic.

Table 14-30. Majority Vote Results

Samples	Vote
000	0
101	1
001	0
111	1

The vote logic captures a sample on every rising edge of BRM_CLK; however, the receiver uses 16x oversampling to take its value in the middle of the sample character.

The receiver starts to count when the start bit is set; however, it does not capture the contents of the RxFIFO at the time the start bit is set. The start bit is validated when zeros are received for 7 consecutive 1/16 of bit times following the 1-to-0 transition. Once the counter reaches 0xF, it starts counting on the next bit and captures it in the middle of the sampling frame as noted in Table 14-30. All data bits are captured in the same manner. Once the stop bit is detected, the receiver shift register (SIPO_OUT) data is parallel shifted to the RxFIFO.

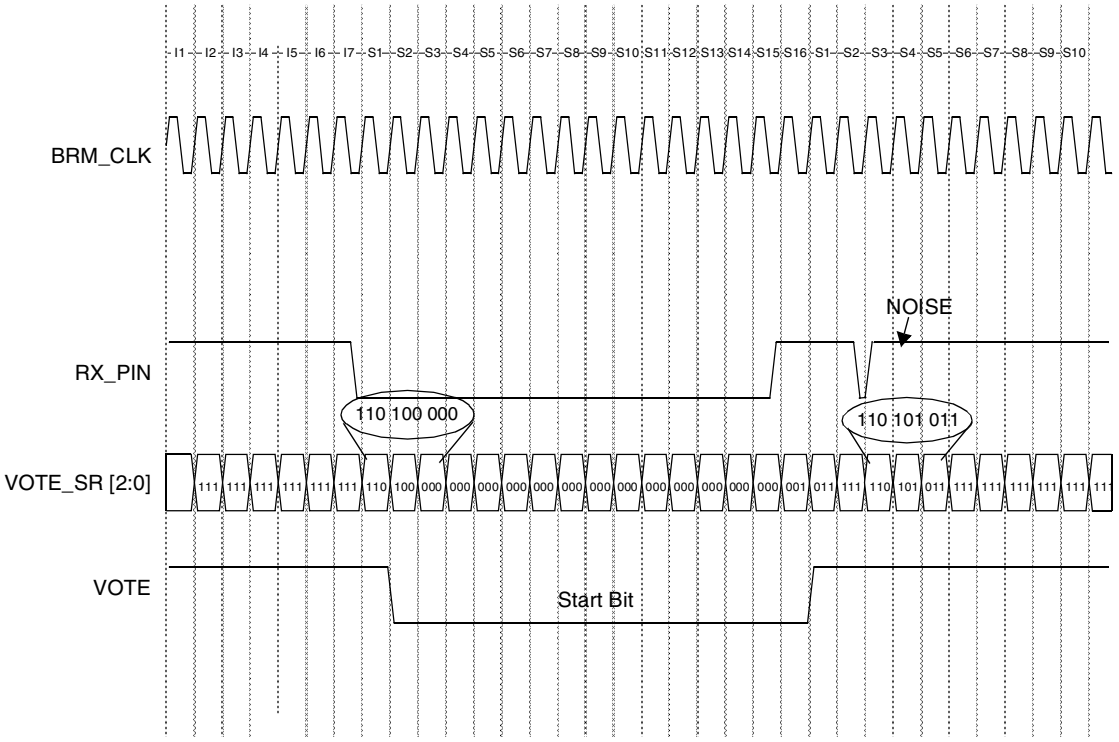


Figure 14-27. Majority Vote Results

A new feature has been recently implemented, which allows resynchronization of the counter on each edge of the RXD line. This is automatic and allows improvement of the immunity of the SIRI against signal distortion.

There is a special case when the BRM_CLK frequency is too low and is unable to capture a 0 pulse in IrDA. In this case, the software must set SCR4[IRSC] so that the reference clock (after internal divider) is used for the voting logic. The pulse is validated by counting the length of the pulse.

14.5.2.8 Binary Rate Multiplier (BRM)

The BRM submodule receives *ref_clk* (*siri_clkin* clock after divider). From this clock, and with integer and non-integer division, the BRM generates all baud rates. The input and output frequency ratio is programmed in the SIRI BRM incremental register (SBIR) and SIRI BRM MOD register (SBMR). The output frequency is divided by the input frequency to produce this ratio. For integer division, set the SBIR = 0x000F and write the divisor to the SBMR register. All values written to these registers must be one less than the actual value to eliminate division by 0 (undefined), and to increase the maximum range of the registers.

Updating the BRM registers requires writing to both registers. The SBIR register must be written to before writing to the SBMR register. If only one register is written to by the software, the BRM continues to use the previous values.

The following examples show how to determine the values that are to be programmed into the SBIR and SBMR for a given reference frequency and desired baud rate. Equation 14-4 can be used to help determine these values:

Frequency and Baud Rate for SBIR and SBMR ***Eqn. 14-4***

$$\text{BaudRate} = \frac{\text{RefFreq}}{\left(16 \times \frac{\text{UBMR} + 1}{\text{UBIR} + 1}\right)}$$

With

reference frequency (Hz): SIRI reference frequency (*siri_clkin* after RFDIV divider)

baud rate (bit/s): Desired baud rate.

Example 14-3. Integer Division ÷ 21

```
Reference Frequency = 19.44 MHz
SBIR = 0x000F
SBMR = 0x0014
Baud Rate = 925.7 kbit/s
```

NOTE

Observe that each value written to the registers is one less than the actual value.

Example 14-4. Non-Integer Division

```
Reference Frequency = 16 MHz
Desired Baud Rate = 920 Kbits/s
```

$$\frac{\text{UBMR} + 1}{\text{UBIR} + 1} = \frac{\text{RefFreq}}{16 \times \text{BaudRate}} = \frac{16 \times 10^6}{16 \times 920 \times 10^3} = 1.087$$

```
Ratio = 1.087 = 1087 / 1000
SBIR = 999 (decimal)= 0x3E7
SBMR = 1086 (decimal)= 0x43E
Non-Integer Division
Reference Frequency = 25 MHz
Desired Baud Rate = 920 kbit/s
Ratio = 1.69837 = 625 / 368
SBIR = 367 (decimal)= 0x16F
SBMR = 624 (decimal)= 0x270
```

Example 14-5. Non-Integer Division

```
Reference Frequency: 30 MHz
Desired Baud Rate = 115.2 kbit/s
Ratio = 16.276043 = 65153 / 4003
SBIR = 4002 (decimal) = 0x0FA2
SBMR = 65152 (decimal) = 0xFE80
```

14.5.2.9 Baud Rate Automatic Detection Logic

When the baud rate automatic detection logic is enabled, the SIRI locks onto the incoming baud rate. To enable this feature, set the automatic detection of baud rate field (SCR1[ADBR]) and write one to SSR2[ADET] to clear it. When ADET=0 and ADBR = 1, the detection starts. Then, once the beginning of start bit (transition from 1-to-0 of RXD) has been detected, the SIRI starts a counter (SBRC) working at reference frequency. Once the end of start bit is detected (transition from 0-to-1 of RXD), the value of SBRC–1 is directly copied into SBMR register. The SBIR register is filled with 0x000F.

So, at the end of start bit, registers get following values:

```
SBRC = number of reference clock periods (after divider) during start bit.
SBIR = 0x000F
SBMR = SBRC - 1
```

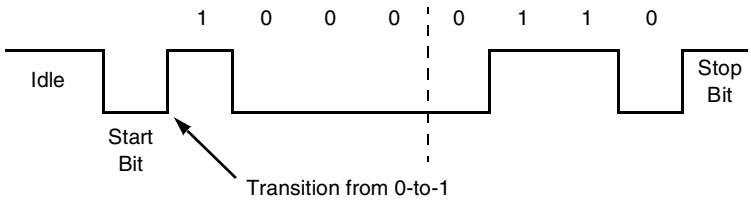
The updated values of the three registers can be read.

Table 14-31 shows parameters for baud rate detection and Figure 14-28 shows the baud rate detection protocol diagram.

If any of the SIRI BRM registers are simultaneously written by the baud rate automatic detection logic and by the peripheral data bus, the peripheral data bus would have higher priority.

Table 14-31. Baud Rate Automatic Detection

ADBR	ADET	Baud Rate Detection	Interrupt
0	X	Manual Configuration	1
1	0	Auto Detection Started	1
1	1	Auto Detection Complete	0
Note: This table assumes that no other interrupt is set at the same time this interrupt is set.			



Note: LSB transmitted first.

Figure 14-28. Baud Rate Detection Protocol Diagram

14.5.2.9.1 Baud Rate Automatic Detection Protocol

The receiver must receive an ASCII character “A” or “a” to verify proper detection of the incoming baud rate. When an ASCII character “A” (0x41) or “a” (0x61) is received and no error occurs, the automatic

detect baud rate bit is set (ADET=1). And if the interrupt is enabled (ADEN=SCR1[15]=1), an interrupt is generated.

When an ASCII character “A” or “a” is not received (because of a bit error or the reception of another character), the auto detection sequence restarts and waits for another 1-to-0 transition.

As long as ADET = 0 and ADBR = 1, the SIRI continues to try to lock onto the incoming baud rate. Once the ASCII character “A” or “a” is detected and the ADET bit is set, the receiver ignores the ADBR bit and continues normal operation with the calculated baud rate.

The SIRI interrupt is active as long as ADET = 1 and ADBR = 1. This can be disabled by clearing the automatic baud rate detection interrupt enable bit (ADEN = 0). Before starting an automatic baud rate detection sequence, set ADET = 0 and ADBR = 1.

The RxFIFO must contain the ASCII character “A” or “a” following the automatic baud rate detection interrupt.

The 16-bit SIRI baud rate count register (SBRC) is reset to 4 and stays at 0xFFFF when an overflow occurs. The SBRC register counts (measures) the duration of start bit. When the start bit is detected and counted, the SBRC retains its value until the next automatic baud rate detection sequence is initiated.

The SBRC counts only when auto detection is enabled.

14.5.2.9.2 Baud Rate Automatic Detection Protocol Improved[1]

New Baud Rate Determination

To reduce distortion and noise on the RXD line, the duration of the baud rate measurement has been extended. Previously, as described above, this determination was based on the measurement of the START bit duration. Now, this measurement is based on the duration of START bit + bit0. Bit0 is the first bit following the START bit. The counter that is started at the falling edge of START bit is no longer stopped at the next rising edge (end of START bit), it is stopped at the next falling edge (end of bit 0). As the character sent is always an “A” (41h) or an “a” (61h), this second falling edge is always present and it indicates the end of bit 0. Once this counter is stopped, the result is divided by 2 and used by the BRM to determine the incoming baud rate.

NOTE

The SBRC register contains the result of this division by two and reflects the measurement of the duration of one bit.

1.Several issues have been reported for ICs using the existing autobaud protocol, especially for 57.6 Kbit/s and 115.2 Kbit/s. As a consequence, this protocol has been improved. The old one is still available in the current SIRI IP, but several modifications can also be used to make this autobaud detection more reliable. If the user wants to keep with the old method, he has to set the bit ADNIMP (SCR3[7]) to 1. If this bit is not set (default), the autobaud improvements protocol is used. Those improvements are grouped in two categories: the new baud rate measurement and the new ACST bit (and associated interrupt).

New Autobaud Counter Stopped Bit and Interrupt

A new bit has been added in SSR2 register: ACST (SSR2[11]). This bit is set immediately after the determination of the baud rate.

- If ADNIMP is not set (default), ACST is set to 1 after the end of bit 0,
- If ADNIMP is set to 1, ACST is set to 1 at the end of START bit.

If ACIEN (SCR3[0]) is set to 1, ACST flags an interrupt. This interrupt informs the core that the BRM has just been set with the result of the bit length measurement. If needed, the core can perform a read of SBMR (or SBRC) register and determine the baud rate measured. Then the core has the possibility to correct the BRM registers with the nearest standardized baud rate.

NOTE

- ACST is set only if ADBR is set to 1, for example, the SIRI is autobauding.
- Clear the ACST bit by writing 1 to it. Writing 0 to the ACST bit has no effect.

14.5.2.10 Escape Sequence Detection

An escape sequence typically consists of three characters entered in rapid succession (such as +++). Because these are valid characters themselves, the time between characters determines if the escape sequence is valid. Too much time between two of the “+” characters is interpreted as two “+” characters, and not part of an escape sequence.

The software chooses the escape character and writes its value to the SIRI escape character register (SESC). The software must also enable escape detection feature by setting SCR2[ESCEN]). The hardware compares this value to incoming characters in the RxFIFO. When an escape character is detected, the internal escape timer starts to count. The software specifies a time-out value for the maximum allowable time between two successive escape characters. Table 14-32 lists these times. The escape timer is programmable in intervals of 2 ms to a maximum interval of 8.192 s.

Table 14-32. Escape Timer Scaling

STIM Register	Maximum Time Between Specified Escape Characters
0x000	2 ms
0x001	4 ms
0x002	6 ms
0x003	8 ms
0x004	10 ms
…	…
0F8	498 ms
0F9	500 ms
…	…

Table 14-32. Escape Timer Scaling (continued)

STIM Register	Maximum Time Between Specified Escape Characters
9C3	5 s
…	…
FFD	8.188 s
FFE	8.190 s
FFF	8.192 s
Note: To calculate the time interval: (STIM_Value + 1) × 0.002 = Time_Interval Example: (09C3 + 1) × 0.002 = 5 s.	

The escape sequence detection feature is available for all the reference frequencies. Before using the escape sequence detection, the user must fill the SONEMS register. This 16-bit register must contain the value of the SIRI internal frequency divided by 1000. The internal frequency is obtained after the SIRI internal divider, which is applied on *siri_clkin*.

The following is an example of a calculation of frequency:

- The SIRI BRM (*siri_clkin*) input clock frequency is 66.5 MHz.
- The SIRI BRM input clock is divided by 2 with the internal divider: SFCR[9:7] = 3’b100.

Calculation of Frequency for SONEMS Register ***Eqn. 14-5***

$$ONEMS = \frac{66.5 \times 10^{6}}{2 \times 1000} = 33250 = 81E2h$$

The escape sequence detection interrupt is triggered when the escape sequence interrupt enable (ESCI) bit is set and an escape sequence is detected (ESCF set). Clear ESCF by writing one to it; writing zero to ESCF has no effect.

14.5.3 FIRI Operation

FIRI is divided into four functional parts: transmitter, receiver, FIFO, and IP interface. The transmitter includes the packet assembler, CRC encoder, and MIR modulator blocks. The receiver includes the searcher, CRC encoder, decoder, and demodulator blocks. The FIRI block diagram is shown in Figure 14-29.

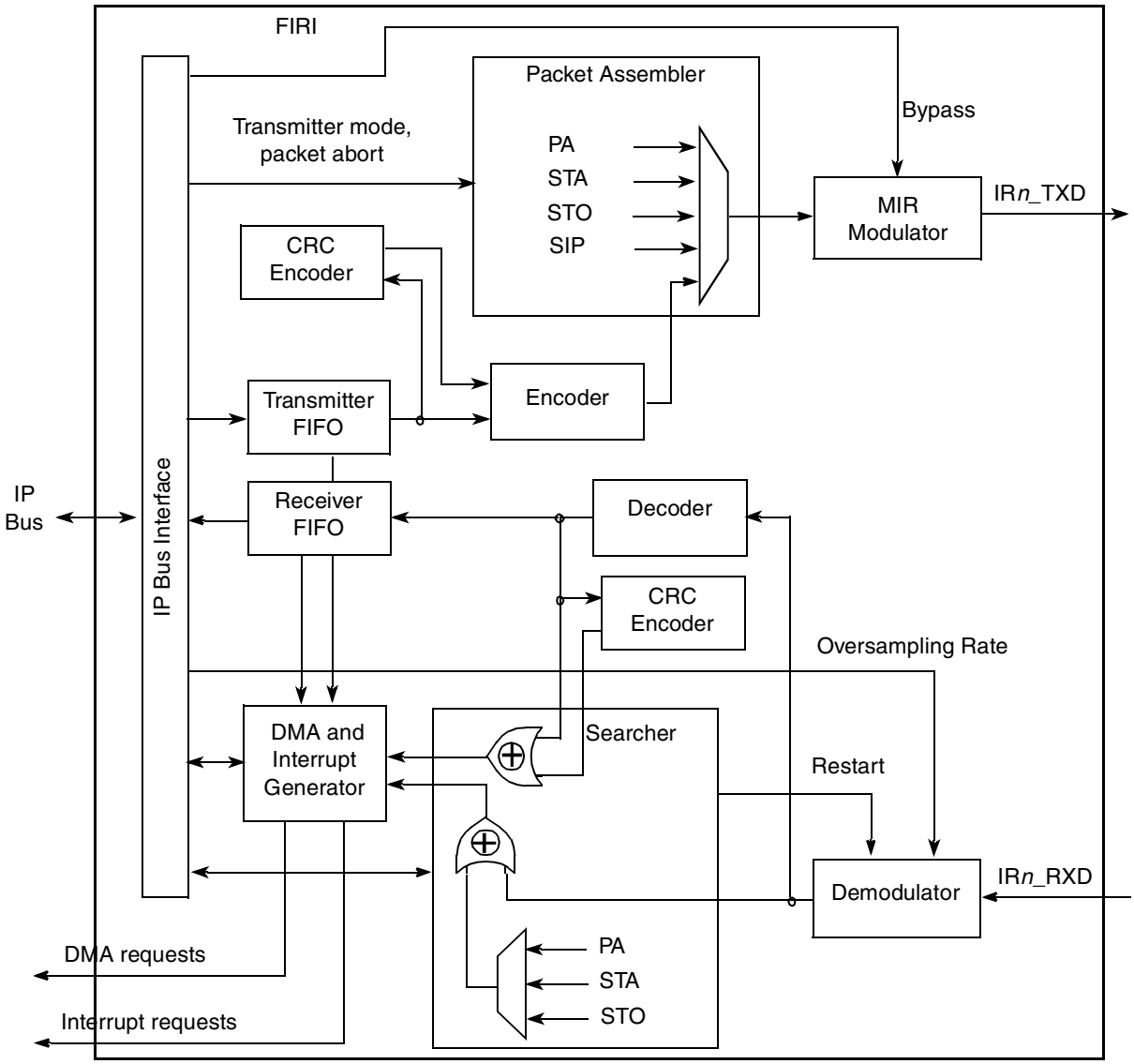


Figure 14-29. FIRI Block Diagram

14.5.3.1 Transmitter Overview

The FIRI transmitter has two modes of operation: MIR and, FIR modes.

14.5.3.1.1 MIR Mode

In MIR mode, the packet assembler block sends the predefined STA sequence (two or more times) to the MIR modulator block. Then the encoder block is involved. The encoder block accesses the FIFO, aligns the data, and sends DD (address, control and data) to the packet assembler block. The CRC field calculated by the CRC block follows the DD field. The encoder block inserts a zero bit after five consecutive ones for DD and CRC fields. After DD and CRC fields the STO bits are sent to the modulator block by the

packet assembler block. The MIR modulator block converts the bit stream to 1/4-bit-length pulses and sends them to the LED.

14.5.3.1.2 FIR Mode

In FIR mode, the MIR modulator block is disabled. The packet assembler block sends the predefined PA (16 or more times) and STA sequences to the LED. Then the encoder block accesses the FIFO, encodes the data (4PPM), and sends DD (address, control and data) to the packet assembler block. The CRC32 field calculated by CRC block follows the DD field after which the STO bits are sent to the modulator.

If the DMA request is not served during MIR or FIR mode and there is no valid data to transmit, the assembler module terminates the packet using the packet abort symbol or with CRC/CRC32 and STO fields (see also PCF bit description) and TFUI interrupt is generated. The packet can also be aborted by the packet length counter or by software (see description of PC bit).

14.5.3.1.3 Serial Infrared Interaction Pulse

At least each 500 ms, the transmitter must send the serial infrared interaction pulse (SIP), in order to guarantee that low-speed IR devices will not interfere. The pulse must be 8.7 µsec wide; the positive phase must be 1.6 µsec wide.

14.5.3.2 Receiver Overview

The FIRI receiver has two modes of operation: MIR and FIR modes. In both modes each incoming pulse is oversampled by the programable factor in the demodulator block. Next, the demodulator block detects the edges of the pulses and synchronizes to the phase of the transmitted pulse. Phase correction is done until the end of the packet because the bit rate tolerance accumulates to very large values for long packets.

14.5.3.2.1 MIR Mode

In MIR mode, the searcher block searches for the STA field. If the sequence is matched, the data stream following the STA field is sent to the decoder block. The decoder block searches for five consecutive ones and skips next the “zero” bit inserted by transceiver for phase synchronization. The data is then written to the FIFO. In parallel, the searcher module searches for the STO field, and when the STO field is detected, a PEI interrupt is generated together with DMA request and the corresponding flag is set in the status register. The searcher module continuously searches for illegal symbols (seven or more consecutive ones). If an illegal symbol is detected, the PAI interrupt is generated and the corresponding flag is set in the status register.

14.5.3.2.2 FIR Mode

In FIR mode, the searcher module searches for the PA field and then for the STA field. If found, the “chip” stream is converted to a data stream by the decoder block and then it is written to the FIFO. In a parallel operation, the searcher module searches for the STO field. When the STO field is detected, a PEI interrupt can be generated together with DMA request. While receiving the searcher module continuously searches the PA, STA, DD, CRC32, and STO fields for illegal symbols. In addition, it searches for a packet abort symbol. On detection of an illegal symbol, the PAI interrupt is generated and the corresponding flag is set in status register.

In MIR and FIR modes the address bits can be compared to a predefined value or/and to broadcast value 0xFF (see description of RAM bits). If CRC field value is does not match the an expected value calculated by the CRC encoder block, a PAI interrupt is generated. The CRC field will be send to the FIFO without regard to the comparison result.

14.5.3.3 FIFOs

The FIRI module has two 128-byte depth FIFOs which are used for transmitter and receiver. The FIFOs operate independently. There are pre-programmed FIFO fill levels which trigger DMA requests. The DMA requests deasserted by the FIFO full and FIFO empty events for transmitter and receiver respectively. If the DMA request was not serviced and, as a result, there is no valid data to transmit or there is no room in the FIFO for received data, the corresponding flags are set in FIRISR. The RFOI and TFUI interrupts are generated (if enabled). The FIFO pointers can only be read by software. The contents of the transmitter or receiver FIFO are lost if the TE or RE bit is cleared. The corresponding FIFO pointers and status bits are cleared as well.

14.5.4 Interrupts and DMA Requests

See Table 14-33 for a list of all interrupts and DMA requests of the SIRI/FIRI. See the register description section for explanation of interrupt enable and status. Register descriptions are contained in Section 14.4.1, “Register Descriptions.”

NOTE

All interrupt types from both infrared controllers map into a single interrupt as presented to the PIC. Software must determine which controller was the interrupt source, as well as the type of interrupt, by polling.

Table 14-33. Interrupts and DMA

Event Type	Interrupt Enable	Interrupt Flag
Receive Interrupts	SCR1[RRDYEN] SCR1[IDEN] SCR2[ATEN] SCR4[DREN] SCR3[RXDSEN] FIRIRCR[RPEIE] FIRIRCR[PAIE] FIRIRCR[RFOIE]	SSR1[RRDY] SSR2[IDLE] SSR1[AGTIM] SSR2[RDR] SSR1[RXDS] FIRIRSR[RPE] FIRIRSR[DDE] FIRIRSR[CRCE] FIRIRSR[RFO]
Transmit Interrupts	SCR1[TXMPTYEN] SCR1[TRDYEN] SCR4[TCEN] FIRITCR[TCIE] FIRITCR[TPEIE] FIRITCR[TFUIE]	SSR2[TXFE] SSR1[TRDY] SSR2[TXDC] FIRITSR[TC] FIRITSR[TPE] FIRITSR[SIPE] FIRITSR[TFU]

Table 14-33. Interrupts and DMA (continued)

Event Type	Interrupt Enable	Interrupt Flag
Other Interrupts	SCR1[ADEN] SCR2[ESCI] SCR3[ACIEN] SCR3[AIRINTEN] SCR3[AWAKEN] SCR3[FRAERREN] SCR3[PARERREN] SCR3[DTREN] (DCE) SCR3[DTRDEN] SCR4[ENIRI] SCR4[OREN] SCR4[BKEN] SCR4[WKEN]	SSR2[ADET] SSR1[ESCF] SSR2[ACST] SSR1[AIRINT] SSR1[AWAKE] SSR1[FRAMEERR] SSR1[PARITYERR] SSR2[DTRF] SSR1[DTRD] SSR2[IRINT] SSR2[ORE] SSR2[BRCD] SSR2[WAKE]
Receive DMA Requests	SCR1[RXDMAEN] SCR1[ATDMAEN] SCR4[IDDMAEN]	SSR1[RRDY] SSR1[AGTIM] SSR2[IDLE]
Transmit DMA Request	SCR1[TXDMAEN]	SSR1[TRDY]

14.6 Initialization/Application Information

14.6.1 Programming the SIRI Interface

14.6.1.1 SIRI High Speed

As an example, the following sequence can be used to program the IrDA interface to send and receive characters at 115.2 Kbit/s.

Assumptions:

- *siri_clkin* = 90 MHz.
- Internal clock divider = 3 (divide *siri_clkin* by 3).
- Baud rate = 115.2 Kbit/s.
- IrDA transceiver is not inverting on both channels: for Tx and Rx, a ‘0’ is represented by a positive pulse, and a ‘1’ is represented by no pulse. (The line stays low.)
- Interrupt: Sent to the core when 1 character is received into the RxFIFO (RDR).

Register values and programming orders:

```
SCR1 = 0x0000_0081
         IREN = 1: Enable IR interface
         SIRIEN = 1: Enable SIRI

UTS = 0x0000

SFCR = 0x0000_0981
         TXTL = 0x02: Default value
         RFDIV = 0x3: Divide siri_clkin by 3 (resulting internal clock is 30 MHz)
```

```
         RXTL = 0x01: Default value

Baud rate = 115.2 kbit/s with internal clock = 30 MHz
SBIR = 0x0000_0202
SBMR = 0x0000_20BE

SCR2 = 0x0000_4027
         STXEN = 1: Enable SIRI transmitter
         WS = 1: Characters are 8-bit length
         TXEN = 1: Enable Rx path
         RXEN = 1: Enable Tx path
         SRST_B = 1: No software reset

SCR3 = 0x0000_0700: Default value
SCR4 = 0x0000_8201
         INVR = 1: Inverted Infrared Reception (because IrDA transceiver is not inverting)
         DREN = 1: To enable RDR interrupt (sent when one char is received)
```

The SIRI is ready to send a character as soon as there is a write into the STXD register. And an interrupt is sent to the core when a character is received.

14.6.1.2 SIRI Low Speed

This time, the assumptions are the same as for high speed, but the speed is now 9.6 Kbit/s.This baud rate is below the limit (even with a minimum pulse duration of 2.5 µs; thus, IRSC and REF30 must be set to 1. Assumptions:

- *siri_clkin* = 90 MHz.
- Internal clock divider = 3 (divide *siri_clkin* by 3).
- Baud rate = 9.6 Kbit/s.
- IrDA transceiver is not inverting on both channels: for Tx and Rx, a ‘0’ is represented by a positive pulse, and a ‘1’ is represented by no pulse. (The line stays low.)
- Interrupt: Sent to the core when 1 character is received into the RxFIFO (RDR).

Register values and programming orders:

```
SCR1 = 0x0000_0081
         IREN = 1: Enable IR interface
         SIRIEN = 1: Enable SIRI

SFCR = 0x0000_0981
         TXTL = 0x02: Default value
         RFDIV = 0x3: Divide siri_clkin by 3 (resulting internal clock is 30 MHz)
         RXTL = 0x01: Default value

Baud rate = 9.6 kbit/s with internal clock = 30 MHz
SBIR = 0x0000_00FF
SBMR = 0x0000_C354


SCR2 = 0x0000_4027
         STXEN = 1: Enable SIRI transmitter
         WS = 1: Characters are 8-bit length
```

```
        TXEN = 1: Enable Rx path
        RXEN = 1: Enable Tx path
        SRST_B = 1: No software reset

SCR3 = 0x0000_0704
        REF30 = 1: Internal SIRI clock = 30 MHz

SCR4 = 0x0000_8221
        INVR = 1: Inverted Infrared Reception (because IrDA transceiver is not inverting)
        IRSC = 1: Because data rate is below the limit and thus the SIRI internal clock
        is used to measure the pulse duration.
        DREN = 1: To enable RDR interrupt (sent when one char is received)
```

The SIRI is now ready to send a character as soon as there is a write into the STXD register. An interrupt is sent to the core when a character is received.

14.6.2 Programming the FIRI Interface

The transmitter can be disabled in the middle of the transfer, see also description of TE bit. However a recommended way to disable the transmitter is to wait till ipi_int_tci_b interrupt, clear all status bits and then clear TE bit.

The receiver behaves differently upon detection of illegal and abort symbols. The receiver is not fill the FIFO by the data following the abort symbol and is searching for the beginning of new packet. In FIR mode the byte previous to abort symbol is not written to the FIFO.

The registers should be accessed with 4 byte width access only. The FIFOs can be access with 1, 2, 4 bytes width accesses. The FIFOs can be accessed by DMA and by core as well. It is strictly not recommended to change the FIFO access width dynamically. This can be done under numerous complicated software restrictions. If it is necessary to change the access width it is recommended to disable the receiver/transmitter, re-program the DMA and then enable the receiver/transmitter again. The width of core’s accesses should be equal to width of DMA accesses.

14.6.2.1 Software Restrictions

- The value of the FIRITCR, FIRITCTR, and FIRICR registers, except the TDT field, should not be changed if the TE field is set or if transmission of the current packet is not completed.
- The value of the FIRIRCR register, except the RDT field, should not be changed if the RE bit is set.
- The write-one-to-clear status bits can be cleared (by software) after a period grater than the chip period from a time point when it was set (by hardware). This restriction can be especially actual when the status bit is cleared at the beginning of interrupt routine.
- The “clear” request of write-one-to-clear status bits is accepted after a period grater then 2 chip periods.
- The write-one-to-clear status bits can not be cleared if the TE/RE bit is cleared.
- If the TE bit is cleared in the middle of a transfer, the transmitter should not be enabled again if the transmission of the current packet is not completed.

- If the RE is cleared by software, it can be set after a period grater then 4 chip periods.

14.6.2.2 Examples of FIRI Programming

14.6.2.2.1 Transmitter Programming Scenario

Goal: to transmit 1024 bytes in FIR mode via infrared link. Packet length should be 512 bytes.

- Configure clock controller. The frequency of IR_CLKIN is 48 MHz (for this example).
- Configure the DMA module for DMA to FIRI transactions. A DMA transaction will be initiated by the negative edge of DMA_DREQ1. The DMA access size is 4 bytes; burst length is four accesses (for this example).
- Configure FIRI to work with 32 bytes burst (FIRICR[BL]). Set FIRI oversampling factor (FIRICR[OSF]) to 6 in order to achieve a 4-Mbit rate with IR_CLKIN.
- Configure FIRI to transmit 512-byte packets (FIRITCTR[TPL]).
- Set the DMA trigger level to 16 (FIRITCR[TDT]). A DMA request will be generated when there are only 16 bytes remain in the FIRI FIFO. Select FIR mode (FIRITCR[TM]).
- Enable the transmitter (FIRITCT[TE]). FIRI immediately asserts a DMA request because the FIFO is empty. The DMA controller delivers the first burst to FIRI. FIRI starts the transmission.

14.6.2.2.2 Receiver Programming Scenario

Data is transmitted in FIR mode. An interrupt should be generated at the end of each packet.

- Configure clock controller. The frequency of IR_CLKIN is 48 MHz (for this example).
- Configure the DMA module for FIRI to DMA transactions. A DMA transaction will be initiated by the negative edge of DMA_DREQ0. The DMA access size is 4 bytes; burst length is four accesses (for this example).
- Configure FIRI to work with 32 bytes burst (FIRICR[BL]). Set FIRI oversampling factor (FIRICR.OSF]) to 6 in order to achieve a 4-Mbit rate with IR_CLKIN.
- Set the DMA trigger level to 16 (FIRIRCR[RDT]). A DMA request will be generated when there are 16 bytes in the FIRI FIFO. Select FIR mode (FIRIRCR[RM]). Enable packet end interrupt.
- Enable the receiver (FIRIRCR[RE]). The receiver searches for PA and STA fields. When a PEI interrupt is generated the software should clear FIRIRSR[RPE].

Chapter 15
Serial Peripheral Interface

15.1 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 15-1, giving an effective FIFO size (latency) of 2 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.

15.2 Introduction

The SPI block diagram is shown in Figure 15-1.

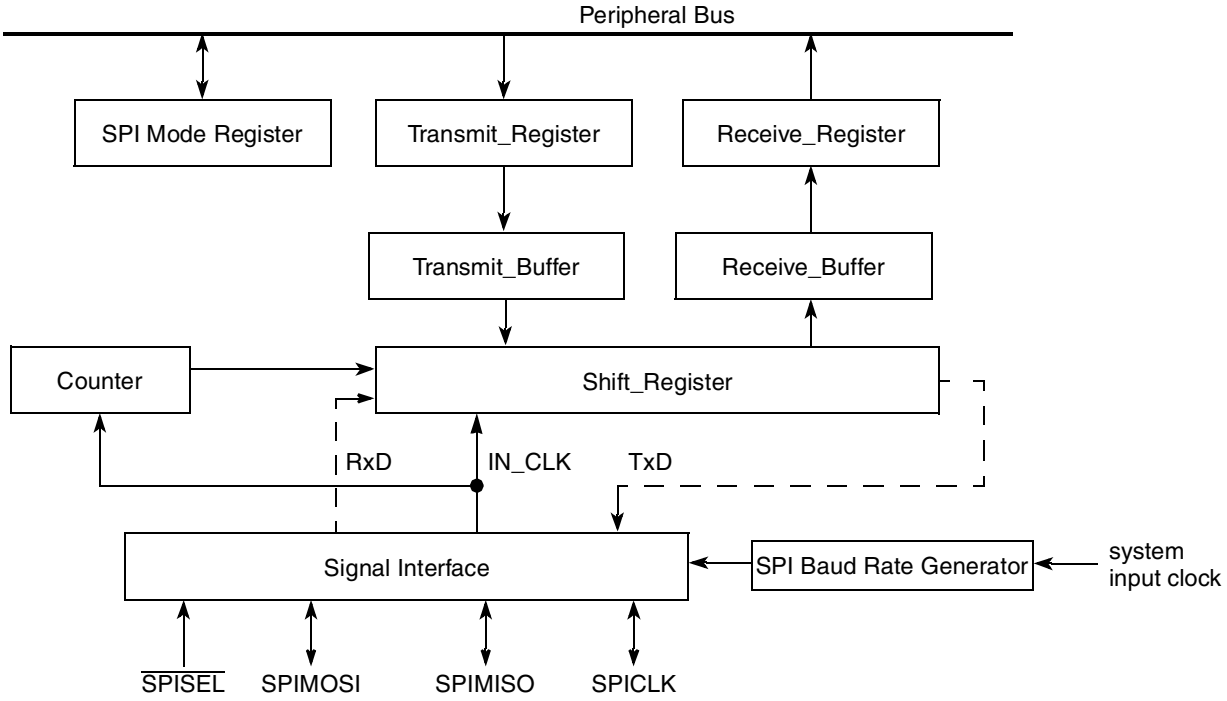


Figure 15-1. SPI Block Diagram

NOTE

On the MPC8610, the SPI block is clocked by the platform clock/2. Throughout this chapter the term “input clock” refers to the platform clock/2.

15.2.1 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

15.2.2 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 15.4.1.4, “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 15-4 for more information.

15.2.3 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.

15.2.3.1 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 15-2. To eliminate the multi-master error in a single-master environment, the master’s $\overline{\text{SPISEL}}$ input should be forced inactive by an external pull up.

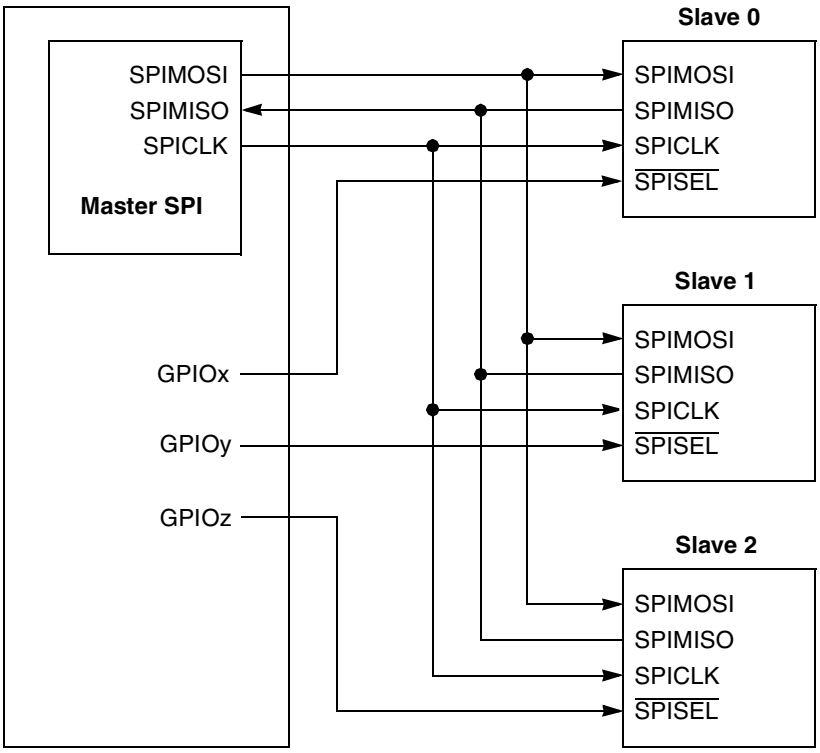


Figure 15-2. Single-Master/Multi-Slave Configuration

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.

15.2.3.2 SPI as a Slave Device

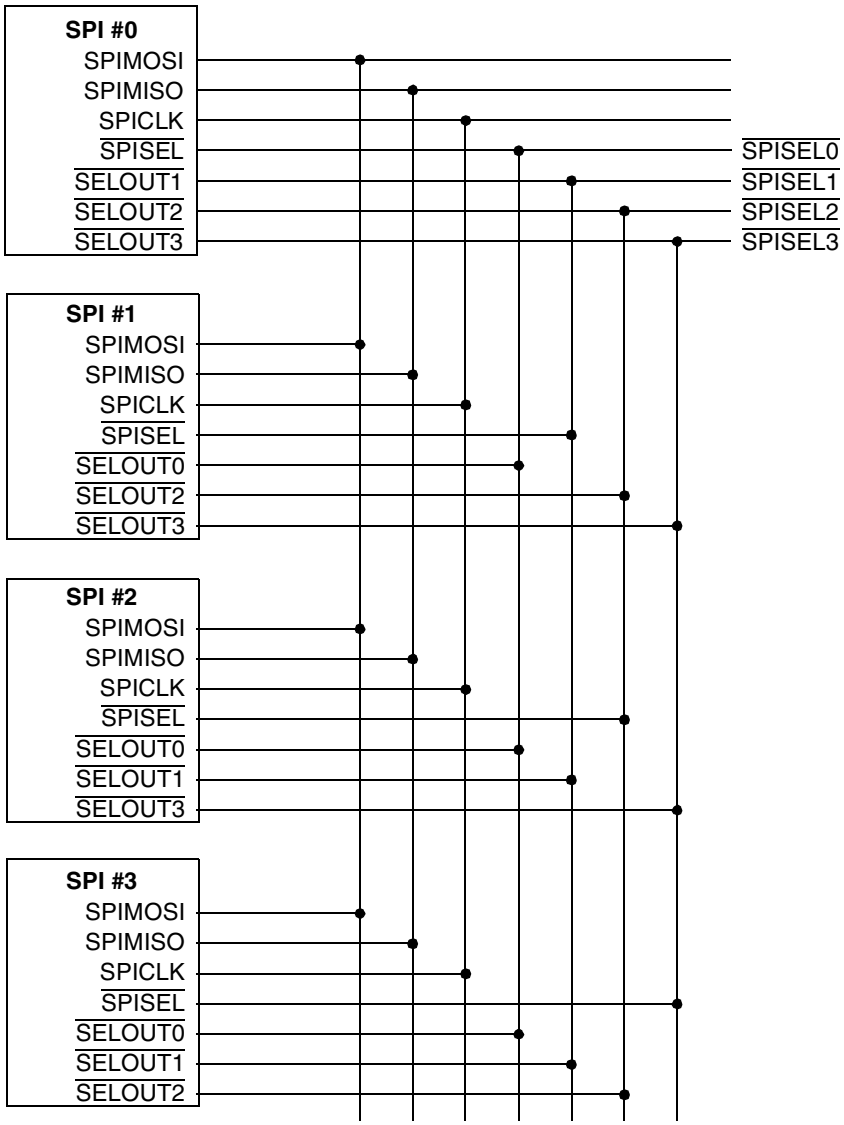
In slave mode, the SPI receives messages from an SPI master and sends a simultaneous reply. The slave's $\overline{\text{SPISEL}}$ must be asserted before Rx clocks are recognized. Once $\overline{\text{SPISEL}}$ is asserted, SPICLK becomes an input from the master to the slave. SPICLK can be any frequency from DC to input clock/2.

To prepare for data transfers, the core writes data to be sent into the SPITD register. Once $\overline{\text{SPISEL}}$ 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.

15.2.3.3 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 15-13. 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, $\overline{\text{SPISEL}}$ 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. $\overline{\text{SELOUT}x}$ signals are implemented in software with general-purpose I/O signals.

Figure 15-3. Multiple-Master Configuration

The maximum sustained data rate that the SPI supports is input clock/50. 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.

15.3 External Signal Descriptions

The SPI's four wire interface consists of transmit, receive, clock, and slave select.

NOTE

The SPI signals are multiplexed with GPIO2 signals. See Section 3.2.5, "UART, SPI, and IR2, and GPIO2 Signal Multiplexing," for more information. The functionality of these signals is determined by the general-purpose I/O control register (GPIOCR) in the global utilities block. Note that the GPIOCR defaults to GPIO functionality on the SPI signal pins; the GPIOCR must be initialized to the desired SPI signaling for proper operation.

15.3.1 Overview

Table 15-1 lists signal properties.

Table 15-1. 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

15.3.2 Detailed Signal Descriptions

Table 15-2 describes the signals in detail.

Table 15-2. 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 15-2. 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.

15.4 Memory Map/Register Definition

Table 15-3 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 CCSRBAR together with the SPI block base address and offset listed in Table 15-3. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 15-3. SPI Register Summary

Offset	Register	Access	Reset Value	Section/Page
SPI block base address: 0x0_7000				
0x000–0x01F	Reserved	—	—	—
0x020	SPI mode register (SPMODE)	R/W	All zeros	15.4.1.1/1515-9
0x024	SPI event register (SPIE)	Mixed	All zeros	15.4.1.2/1515-12
0x028	SPI mask register (SPIM)	R/W	All zeros	15.4.1.3/1515-13
0x02C	SPI command register (SPCOM)	W	All zeros	15.4.1.4/1515-14
0x030	SPI transmit register (SPITD)	W	All zeros	15.4.1.5/1515-14
0x034	SPI receive register (SPIRD)	R	0xFFFF_FFFF	15.4.1.6/1515-15
0x038–0xFFF	Reserved	—	—	—

15.4.1 Register Descriptions

15.4.1.1 SPI Mode Register (SPMODE)

SPMODE, shown in Figure 15-4, controls both the SPI operation mode and clock source.

Offset 0x020 Access: Read/write

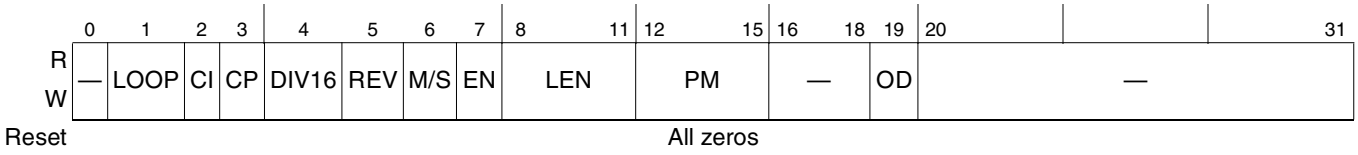


Figure 15-4. SPMODE-SPI Mode Register Definition

Table 15-4 describes the SPMODE fields.

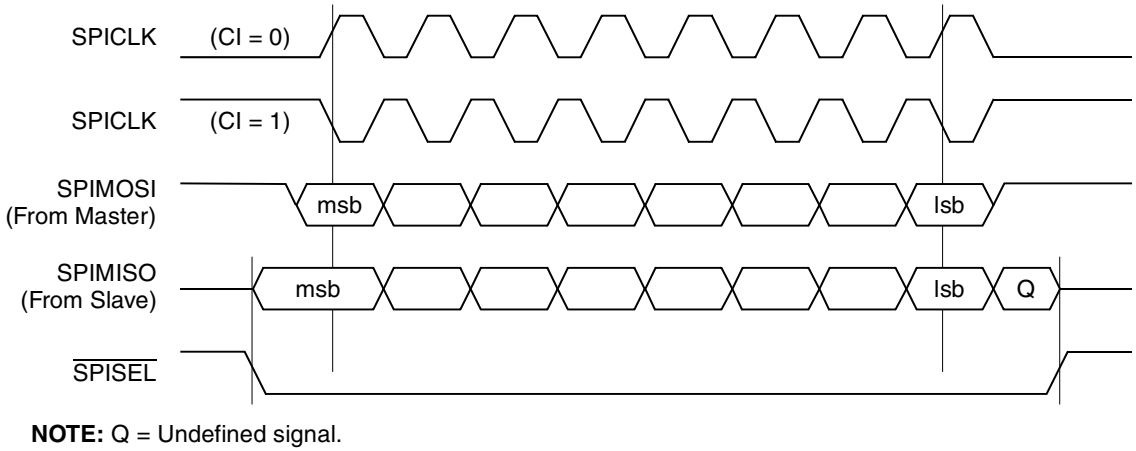
Table 15-4. SPMODE Field Descriptions

Bits	Name	Description
0	—	Reserved
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 an MME error.
2	CI	Clock invert. Inverts SPI clock polarity. See Figure 15-5 and Figure 15-6 for more information. 0 The inactive state of SPICLK is low. 1 The inactive state of SPICLK is high.

Table 15-4. SPMODE Field Descriptions (continued)

Bits	Name	Description
3	CP	Clock phase. Selects the transfer format. See Figure 15-5 and Figure 15-6 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 15.4.1.6.1, “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 $4 \times ([PM] + 1)$, 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 $16 \times (4 \times (0011 + 1)) = 256$. In slave mode this field must be cleared.
16–18	—	Reserved
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	—	Reserved

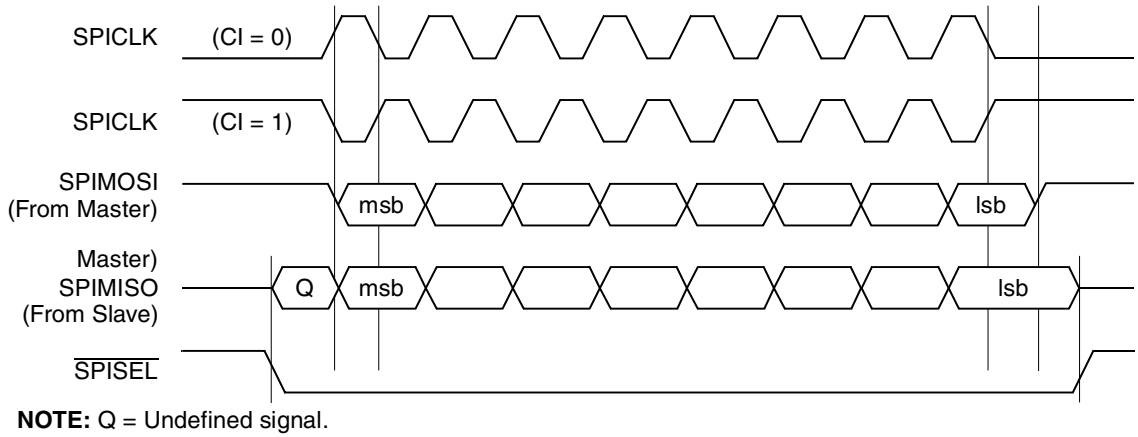
Figure 15-5 shows the SPI transfer format in which SPICLK starts toggling in the middle of the transfer (SPMODE[CP] = 0).



NOTE: Q = Undefined signal.

Figure 15-5. SPI Transfer Format with SPMODE[CP] = 0

Figure 15-6 shows the SPI transfer format in which SPICLK starts toggling at the beginning of the transfer (SPMODE[CP] = 1).



NOTE: Q = Undefined signal.

Figure 15-6. SPI Transfer Format with SPMODE[CP] = 1

15.4.1.2 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 15-7 shows SPI event register.

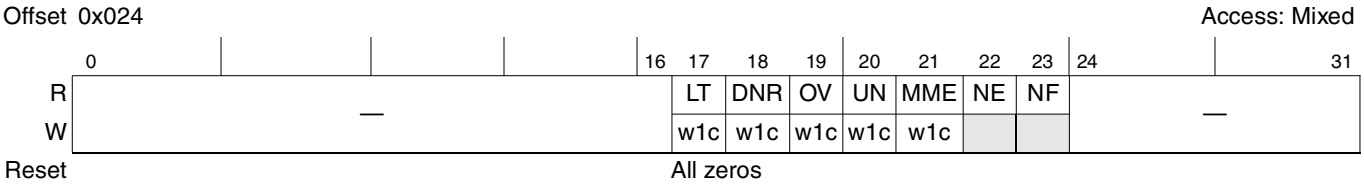


Figure 15-7. SPIE—SPI Event Register Definition

Table 15-5 describes the SPIE fields.

Table 15-5. SPIE Field Descriptions

Bits	Name	Description
0–16	—	Reserved
17	LT	Last character was transmitted. The last character of the frame was completely transferred. This bit is set only if the transmitted character was the last character of the frame (if SPCOM[LST] is set). New data can be written to SPITD is indicated by bit NF.
18	DNR	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. **Note:** DNR capability is not supported on the MPC8610. This bit should be considered reserved.
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

15.4.1.3 SPI Mask Register (SPIM)

The SPI mask register (SPIM), shown in Figure 15-8, 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 15-8. SPIM—SPI Mask Register Definition

Table 15-6 describes the SPIM fields.

Table 15-6. SPIM Field Descriptions

Bits	Name	Description
0–16	—	Reserved
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 **Note:** DNR capability is not supported on the MPC8610. This bit should be considered reserved.
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

15.4.1.4 SPI Command Register (SPCOM)

The SPI command register (SPCOM), shown in Figure 15-9, is used to end SPI operation.

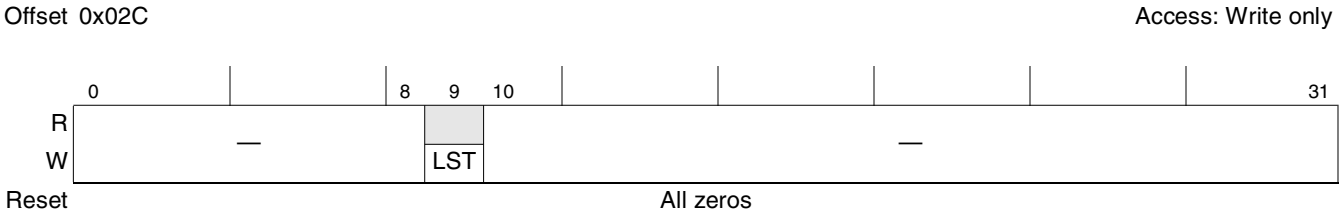


Figure 15-9. SPI Command Register Definition

Table 15-7 describes the SPCOM fields.

Table 15-7. SPCOM Field Descriptions

Bits	Name	Description
0–8	—	Reserved
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

15.4.1.5 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 15-10 shows the SPI transmit data hold register.

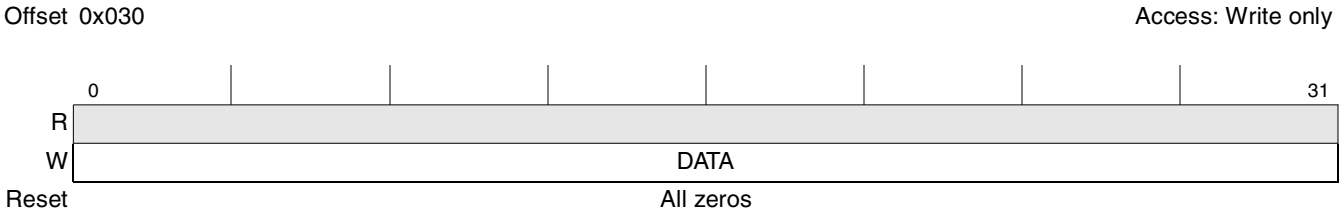


Figure 15-10. SPI Transmit Data Hold Register Definition

Table 15-8 shows the field descriptions of the SPI transmit data hold register.

Table 15-8. SPI Transmit Data Hold Field Descriptions

Bits	Name	Description
0–31	DATA	These bits are the data to be sent.

15.4.1.6 SPI Receive Data Hold Register (SPIRD)

SPIRD, shown in Figure 15-11, is used to receive a character of data from the SPI channel. Each time SPIE[NE] is set, the core can read SPIRD.

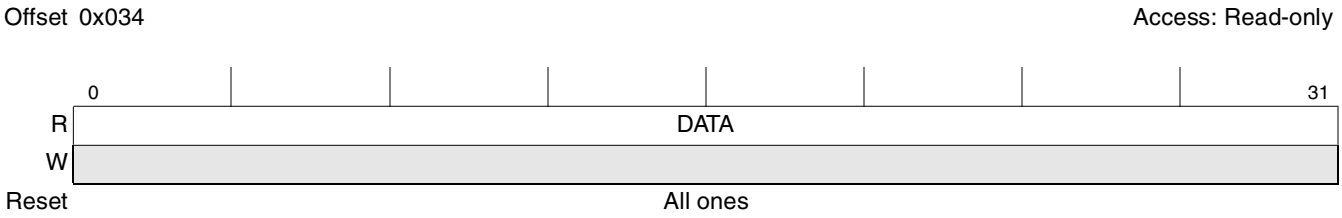


Figure 15-11. SPI Receive Data Hold Register Definition

Table 15-9 shows the field descriptions of the SPI receive data hold register.

Table 15-9. SPI Receive Data Hold Field Descriptions

Bits	Name	Description
0–31	DATA	Received data. These bits are the received data from the SPI bus.

15.4.1.6.1 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).

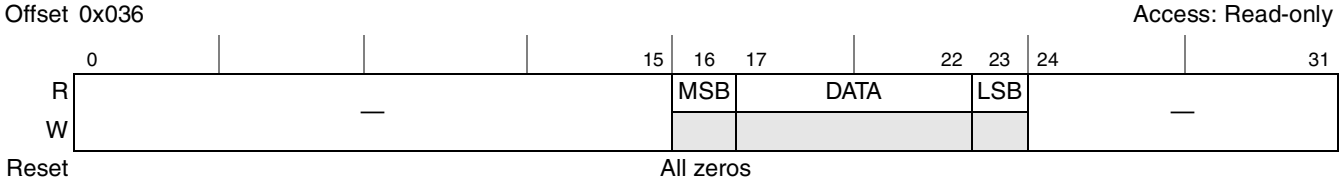


Figure 15-12. Example SPMODE[REV] = 0 SPMODE[LEN] = 7 LSB Sent First

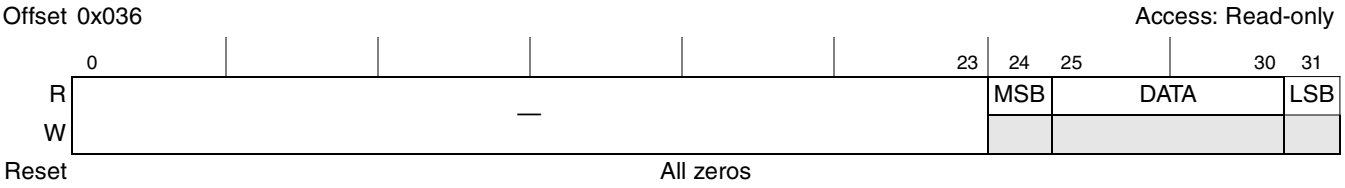


Figure 15-13. Example SPMODE[REV] = 1 SPMODE[LEN] = 7 MSB Sent First

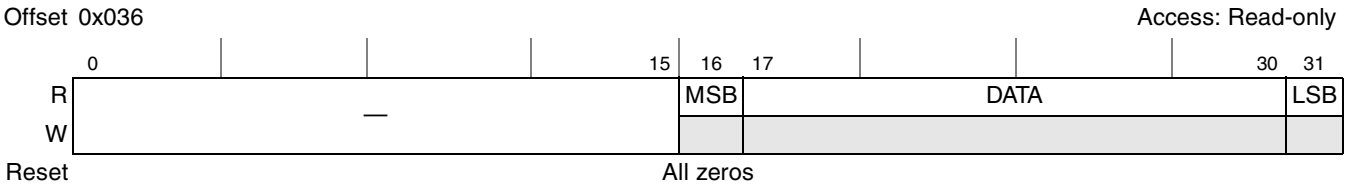


Figure 15-14. Example SPMODE[REV] = 1 SPMODE[LEN] = 15 MSB Sent First

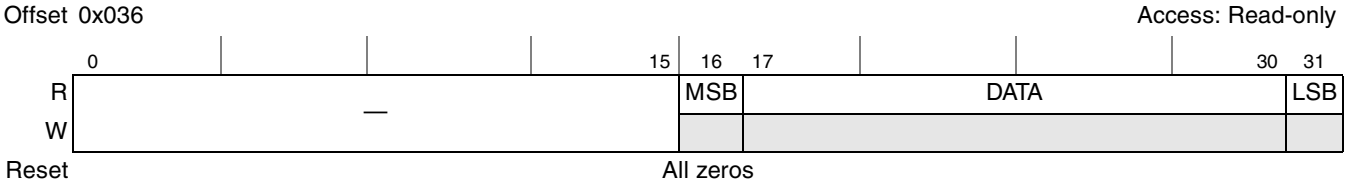


Figure 15-15. Example SPMODE[REV] = 0 SPMODE[LEN] = 15 LSB Sent First

15.5 Initialization/Application Information

The following sections describe programming examples of the SPI master and slave.

15.5.1 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 0xFFFFFFFF 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.

15.5.2 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 0xFFFFFFFF 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.

Chapter 16
Synchronous Serial Interface (SSI)

This chapter describes the synchronous serial interface (SSI) and discusses the architecture, programming model, operating modes, and initialization of the SSI.

See Figure 16-1 for a block diagram of the SSI. It consists of control registers to set up the port, status register, separate transmit and receive circuits with FIFO registers, and separate serial clock and frame sync generation for the transmit and receive sections. The second set of Tx and Rx FIFOs replicates the logic used for the first set.

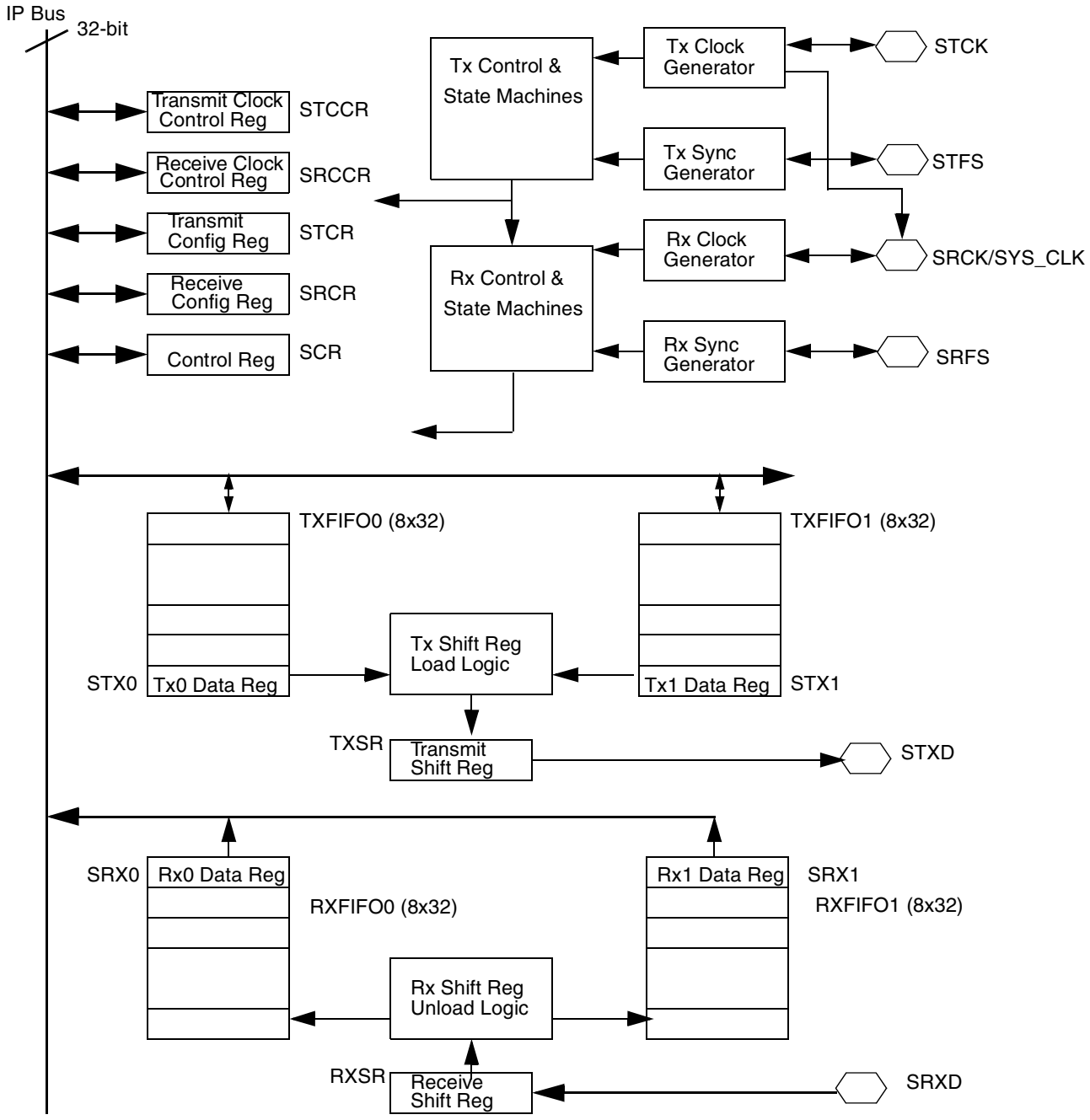


Figure 16-1. SSI Block Diagram

16.1 SSI Overview

The SSI is a full-duplex, serial port that allows the chip to communicate with a variety of serial devices. These serial devices can be standard CODer-DECoder (CODECs), digital signal processors (DSPs), microprocessors, peripherals, and popular industry audio CODECs that implement the inter-IC sound bus standard (I^2S) and Intel AC97 standard.

SSI is typically used to transfer samples in a periodic manner. The SSI consists of independent transmitter and receiver sections with independent clock generation and frame synchronization.

16.1.1 SSI Features

The SSI includes the following features:

- Independent (asynchronous) or shared (synchronous) transmit and receive sections with separate or shared internal/external clocks and frame syncs, operating in Master or Slave mode.
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with as many as thirty-two time slots
- 2 sets of Transmit and Receive FIFOs. Each of the four FIFOs is 8x32 bits. The two sets of Tx/Rx FIFOs can be used in Network mode to provide 2 independent channels for transmission and reception
- Programmable data interface modes such as I2S, lsb, msb aligned
- Programmable word length (8, 10, 12, 16, 18, 20, 22 or 24 bits)
- Program options for frame sync and clock generation
- Programmable I2S modes (Master, Slave or Normal). Oversampling clock is available as output from SRCK in I2S Master mode.
- AC97 support
- Completely separate clock and frame sync selections for the receive and transmit sections. In AC97 standard, the clock is taken from an external source and frame sync is generated internally.
- External SSI clock input for use in I2S Master mode.
- Programmable internal clock divider
- Time Slot Mask Registers for reduced CPU overhead (for Tx and Rx both)
- SSI power-down feature
- Programmable wait states for CPU accesses
- IP Interface for register accesses, compliant to SRS 3.0.2 standard

16.1.2 SSI Modes of Operation

SSI has the following basic operating modes.

- Normal mode
  - — Asynchronous protocol
  - — Synchronous protocol
- Network mode
  - — Asynchronous protocol
  - — Synchronous protocol

These modes can be programmed by several bits in the SSI control registers. See Table 16-1 for the list of SSI operating modes and some of the typical applications in which they can be used:

Table 16-1. SSI Operating Modes

TX, RX Sections	Serial Clock	Mode	Typical Application
Asynchronous	Continuous	Normal	Multiple synchronous CODECs
Asynchronous	Continuous	Network	TDM CODEC or DSP networks
Synchronous	Continuous	Normal	Multiple synchronous CODECs
Synchronous	Continuous	Network	TDM CODEC or DSP network

The transmit and receive sections of the SSI can be synchronous or asynchronous. In Synchronous mode, the transmitter and the receiver use a common clock and frame synchronization signal. The RXBIT0 and RSHFD bits in SRCR still affect shifting-in of received data in synchronous mode. In Asynchronous mode, the transmitter and receiver each has its own clock and frame synchronization signals. In Continuous mode, the clock runs continuously.

Normal or Network mode can also be selected. In Normal mode, the SSI functions with one data word of I/O per frame. In Network mode, any number from two to thirty-two data words of I/O per frame can be used. Network mode is typically used in star time division multiplex networks with other processors or CODECs, allowing interface to time division multiplexed networks without additional logic. These distinctions result in the basic operating modes that allow the SSI to communicate with a wide variety of devices.

The SSI supports both Normal and Network modes, and these can be selected independently of whether the transmitter and receiver are synchronous or asynchronous. Typically these protocols are used in a periodic manner, where data is transferred at regular intervals, such as at the sampling rate of an external CODEC. Both modes use the concept of a frame. The beginning of the frame is marked with a frame sync when programmed with continuous clock. The frame sync occurs at a periodic interval. The length of the frame is determined by the DC[4:0] bits in either the SRCCR or STCCR register, depending on whether data is being transmitted or received. The number of words transferred per frame depends on the mode of the SSI.

In Normal mode, one data word is transferred per frame. In Network mode, the frame is divided into anywhere between two and thirty-two time slots, where in each time slot one data word can optionally be transferred.

Apart from the above basic modes of operation, SSI supports the following modes which require some specific programming.

- I2S mode
- AC97 mode
  - — AC97 Fixed mode
  - — AC97 Variable mode

In (non-I2S) slave modes (external frame sync), the programmed word length setting of the SSI should be equal to the word length setting of the master. In I2S slave mode, the programmed word length setting of the SSI can be lesser than or equal to the word length setting of the I2S master (external CODEC).

In slave modes, the programmed frame length setting (DC bits) of the SSI can be lesser than or equal to the frame length setting of the master (external CODEC).

The following sections provide detailed descriptions of the above modes.

16.1.2.1 Normal Mode

Normal mode is the simplest mode of the SSI. It is used to transfer data in one time slot per frame. A time slot is a unit of data and the WL[3:0] bits define the number of bits in a time slot. In Continuous Clock mode, a frame sync occurs at the beginning of each frame. The length of the frame is determined by the following factors:

- The period of the Serial Bit Clock (DIV2, PSR, PM[7:0] bits for internal clock or the frequency of the external clock on the STCK port)
- The number of bits per time slot (WL[3:0] bits)
- The number of time slots per frame (DC[4:0] bits)

If Normal mode is configured with more than one time slot per frame, data is transferred only in the first time slot. No data is transferred in subsequent time slots. In Normal mode, DC[4:0] values corresponding to more than a single time slot in a frame, only result in lengthening the frame. Data transfer only takes place during the first time slot of the frame.

16.1.2.1.1 Normal Mode Transmit

The conditions for data transmission from the SSI in Normal mode are:

- SSI enabled (SSIEN = 1)
- Enable FIFO and configure Transmit and Receive Watermark if FIFO is used.
- Write data to Transmit Data Register (STX)
- Transmitter enabled (TE = 1)
- Frame sync active (for continuous clock case)

When the above conditions occur in Normal mode, the next data word is transferred into the Transmit Shift Register (TXSR) from the Transmit Data Register 0 (STX0), or from the Transmit FIFO 0 Register, if transmit FIFO 0 is enabled. The new data word is transmitted immediately.

If transmit FIFO 0 is not enabled and the transmit data register empty (TDE0) bit is set, transmit interrupt 0 occurs if the transmit interrupt enable (TIE) and TDE0_EN bits are set.

The Transmit FIFO Empty 0 (TFE0) bit is set if the Transmit FIFO 0 reaches the selected threshold. If transmit FIFO 0 is enabled and the Transmit FIFO Empty (TFE0) bit is set, transmit interrupt 0 occurs if the transmit interrupt enable (TIE) and TFE0_EN bits are set. If transmit FIFO 0 is enabled and filled with data, 8 data words can be transferred before the core must write new data to the STX0 register.

The STXD port is disabled except during the data transmission period. For a continuous clock, the optional frame sync output and clock outputs are not disabled, even if both receiver and transmitter are disabled.

16.1.2.1.2 Normal Mode Receive

The conditions for data reception from the SSI are:

- SSI enabled (SSIEN = 1)
- Receiver enabled (RE = 1)
- Frame sync active (for continuous clock case)

With the above conditions in Normal mode with a continuous clock, each time the frame sync signal is generated (or detected) a data word is clocked in.

If receive FIFO 0 is not enabled, the received data word is transferred from the Receive Shift Register (RXSR) to the Receive Data Register 0 (SRX0), the Receive Data Ready 0 (RDR0) flag is set. Receive Interrupt 0 occurs if RIE and RDR0_EN bits are set.

If receive FIFO 0 is enabled, the received data word is transferred to the Receive FIFO 0. The Receive FIFO Full 0 (RFF0) flag is set if the Receive Data Register (SRX0) is full and Receive FIFO 0 reaches the selected threshold. Receive Interrupt 0 occurs if Receive Interrupt Enable (RIE) and RFF0_EN bits are set.

The core program has to read the data from the Receive Data Register 0 (SRX0) before a new data word is transferred from the Receive Shift Register (RXSR), otherwise the Receive Overrun Error 0 (ROE0) bit is set. If receive FIFO 0 is enabled, the Receive Overrun Error 0 (ROE0) bit is set when the Receive FIFO 0 data level reaches the selected threshold and a new data word is ready to be transferred to the Receive FIFO 0.

See Figure 16-2 for illustration of transmitter and receiver timing for an 8-bit word in the first time slot in Normal mode, continuous clock with a late word length frame sync. The Tx Data register is loaded with the data to be transmitted. On arrival of the clock, this data is transferred to the Transmit Shift Register which gets transmitted on arrival of the frame-sync on the STXD output. Simultaneously, the Receive Shift Register shifts in the received data available on the SRXD input and at the end of the time slot, this data is transferred to the Rx Data Register.

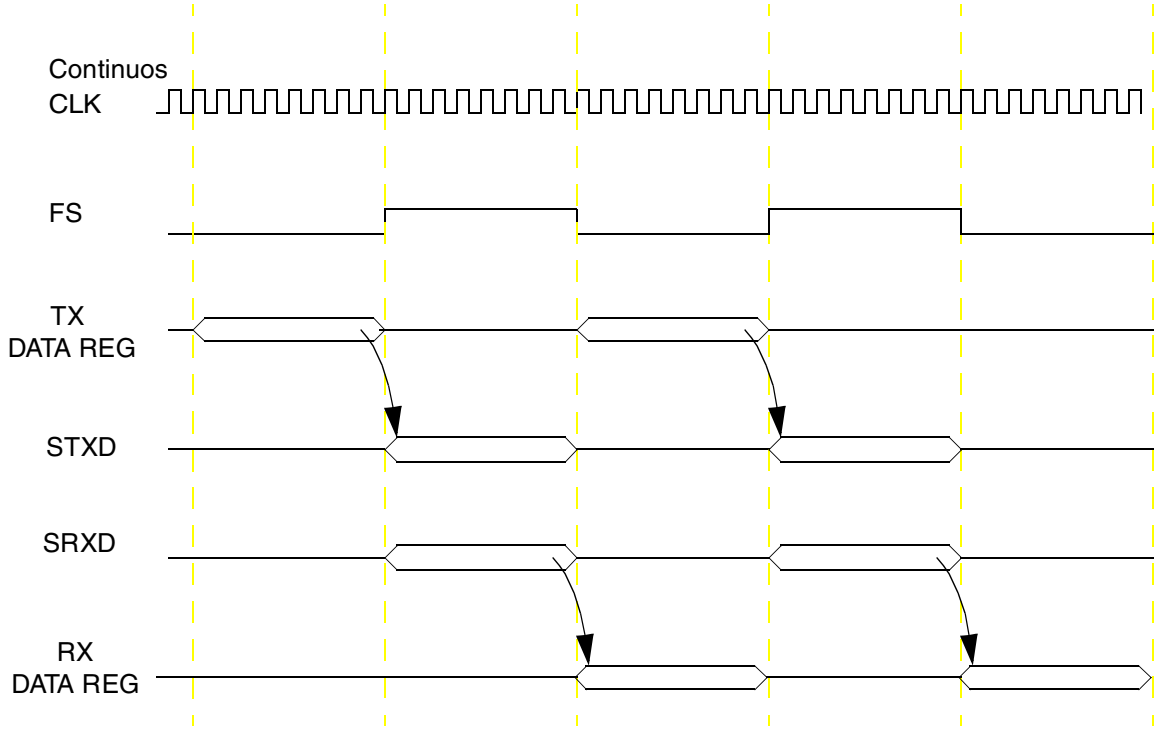


Figure 16-2. Normal Mode Timing - Continuous Clock

16.1.2.2 Network Mode

Network mode is used for creating a Time Division Multiplexed (TDM) network, such as a TDM CODEC network or a network of DSPs. In Continuous Clock mode, a frame sync occurs at the beginning of each frame. In this mode, the frame is divided into more than one time slot. During each time slot, one data word can be transferred. Each time slot is then assigned to an appropriate CODEC or DSP on the network. The DSP can be a master device that controls its own private network, or a slave device that is connected to an existing TDM network and occupies a few time slots.

The frame sync signal indicates the beginning of a new data frame. Each data frame is divided into time slots and transmission and/or reception of one data word can occur in each time slot (rather than in just the frame sync time slot as in Normal mode). The frame rate dividers, controlled by the DC[4:0] bits, select two to thirty-two time slots per frame. The length of the frame is determined by the following factors:

- The period of the serial bit clock (PSR, PM[7:0] bits for internal clock, or the frequency of the external clock on the STCK port)
- The number of bits per sample (WL[3:0] bits)
- The number of time slots per frame (DC[4:0] bits)

In Network mode, data can be transmitted in any time slot. The distinction of the Network mode is that each time slot is identified with respect to the frame sync (data word time). This time slot identification allows the option of transmitting data during the time slot by writing to the STX registers or ignoring the time slot as determined by STMSK register bits. The receiver is treated in the same manner and received

data is only transferred to the receive data register/fifo if the corresponding time slot is enabled (through SRMSK).

By utilizing the STMSK and SRMSK registers, software only has to service the SSI during valid time slots. This eliminates any overhead associated with unused time slots. See Section 16.3.1.18, "SSI Transmit Time Slot Mask Register (STMSK)," and Section 16.3.1.19, "SSI Receive Time Slot Mask Register (SRMSK)," for more information on STMSK and SRMSK.

In the two-channel mode of operation, the second set of Transmit and Receive FIFOs and Data Registers are used to create two separate channels. These channels are completely independent, with a their own set of Core interrupts which are identical to the ones available for the default channel (although they do not have DMA channels). In this mode, data is transmitted/received in enabled time slots alternately from/to FIFO 0 and FIFO 1, starting from FIFO 0. The first data word is taken from FIFO 0 and transmitted in the first enabled time slot and subsequently, data is loaded from FIFO 1 and FIFO 0 alternately and transmitted. Similarly, the first received data is sent to FIFO 0 and subsequent data is sent to FIFO 1 and FIFO 0 alternately. Time slots can be selected through the Transmit and Receive Time Slot Mask registers (STMSK and SRMSK). For using this mode of operation, the TCH_EN bit (SCR[8]) needs to be set.

16.1.2.2.1 Network Mode Transmit

The transmit portion of SSI is enabled when the SSIEN and the TE bits in the SCR are both set. However, for continuous clock, when the TE bit is set, the transmitter is enabled only after detection of a new frame sync (transmission starts from the next frame boundary).

Normal start-up sequence for transmission is to perform the following:

1. Write the data to be transmitted to the STX register. This clears the TDE flag.
2. Set the TE bit to enable the transmitter on the next word boundary (for continuous clock case).
3. Enable transmit interrupts.

Alternatively, the programmer may decide not to transmit in a time slot by configuring the STMSK. The TDE flag is not cleared, but the STXD port remains disabled during the time slot. When the frame sync is detected or generated (continuous clock), the first enabled data word is transferred from the STX register to the TXSR and is shifted out (transmitted). When the STX register is empty, the TDE bit is set, which causes a transmitter interrupt (in case the FIFO is disabled) to be sent if the TIE bit is set. Software can poll the TDE bit or use interrupts to reload the STX register with new data for the next time slot. Failing to reload the STX register before the TXSR is finished shifting (empty) causes a transmitter underrun and the TUE error bit is set. In case the FIFO is enabled, the TFE flag is set in accordance with the watermark setting and this flag causes the transmitter interrupt to occur.

The operation of clearing the TE bit disables the transmitter after completion of transmission of the current frame. Setting the TE bit enables transmission from the next frame. During that time the STXD port is disabled. The TE bit should be cleared after the TDE bit is set to ensure that all pending data is transmitted.

To summarize, the Network mode transmitter generates interrupts every enabled time slot and requires the core program to respond to each enabled time slot. These responses from the core are one of the following:

- Write data in data register to enable transmission in the next time slot.

- Configure the time slot register to disable transmission in the next time slot (unless time slot is already masked by STMSK register bit).
- Do nothing—transmit underrun occurs at the beginning of the next time slot and the previous data is re-transmitted.

In the two-channel mode of operation, both the channels (Data Registers, FIFOs, and interrupts) operate in the same manner, as described above. The only difference in case of the second channel is that the interrupts related to this channel are generated only in case this mode of operation is selected (TDE1 is low by default).

16.1.2.2.2 Network Mode Receive

The receiver portion of the SSI is enabled when both the SSIEN and the RE bits in the SCR are set. However, the receive enable only takes place during that time slot if RE is enabled before the second to last bit of the word. If the RE bit is cleared, the receiver is disabled at the end of the current frame. SSI is capable of finding the start of the next frame automatically. When the word is completely received, it is transferred to the SRX register, which sets the RDR bit (Receive Data Ready). Setting the RDR bit causes a receive interrupt to occur if the receiver interrupt is enabled (the RIE bit is set). The second data word (second time slot in the frame), begins shifting in immediately after the transfer of the first data word to the SRX register. The core program has to read the data from the Receive Data Register (which clears RDR) before the second data word is completely received (ready to transfer to RX data register) or a receive overrun error occurs (the ROE bit is set).

An interrupt can occur after the reception of each enabled data word or the programmer can poll the RDR flag. The core program response can be one of the following:

- Read RX and use the data.
- Read RX and ignore the data.
- Do nothing—the receiver overrun exception occurs at the end of the current time slot.

NOTE

For a continuous clock, the optional frame sync output and clock output signals are not affected, even if the transmitter or receiver is disabled. TE and RE do not disable the bit clock or the frame sync generation. To disable the bit clock and the frame sync generation, the SSIEN bit in the SCR can be cleared, or the port control logic external to the SSI (for example, in the IOMUX) can be reconfigured.

In the two-channel mode of operation, both the channels (Data Registers, FIFOs and interrupts) operate in the same manner, as described above. The only difference in case of the second channel is that the interrupts related to this channel are generated only in case this mode of operation is selected.

The transmitter and receiver timing for an 8-bit word with continuous clock, FIFO disabled, three words per frame sync in Network mode is shown in Figure 16-3.

NOTE

The transmitter repeats the value 0x5E because of an underrun condition

For the transmit section, the STMSK value is updated in the last time slot of frame 1, to mask the first two time slots (0x3). This value takes effect from the next time slot and consequently, the next frame transmits data in the third time slot only.

For the receive section, data received on the SRXD pin gets transferred to the Rx Data register at the end of each time slot. If the FIFO is disabled, the RDR flag gets set and causes a receiver interrupt if RE, RIE and RDR_EN bits are set. If the FIFO is enabled, then the RFF flag is used for interrupt generation (this flag is set in accordance with the watermark settings). Here all time slots are enabled. The receive data ready flag is set after reception of the first data (0x55). Since the flag is not cleared (Rx Data Register is not read by core), the Receive Overrun Error (ROE) flag is set on reception of the next data (0x5E). ROE flag is cleared on writing ‘1’ to the corresponding interrupt status bit in SSI Status Register.

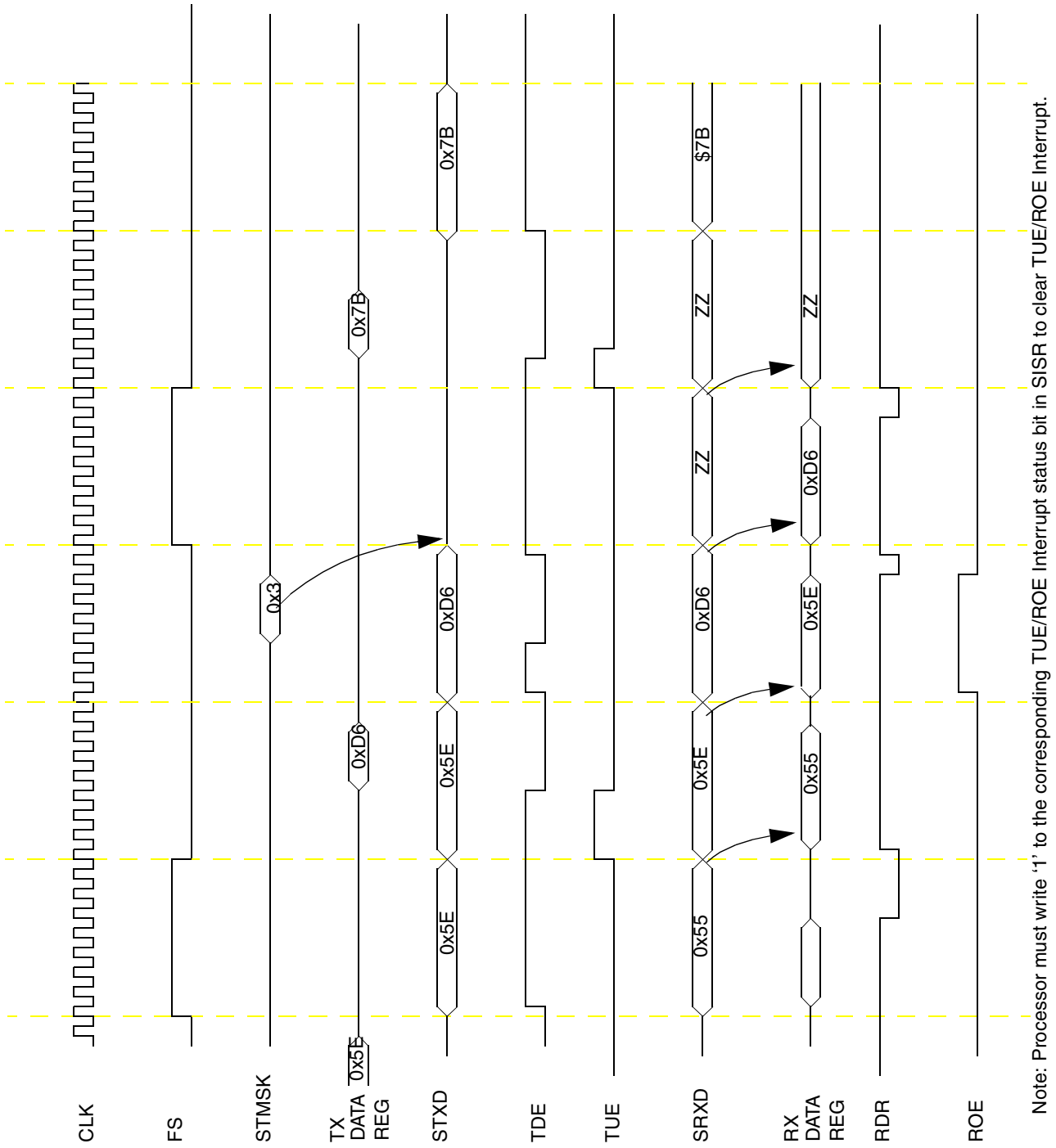


Figure 16-3. Network Mode Timing - Continuous Clock

16.1.2.3 I2S Mode

The SSI is compliant to I^2S bus specification from Philips Semiconductors (February 1986, Revised June 5, 1996). See Figure 16-4 for an illustration of the basic I2S protocol timing.

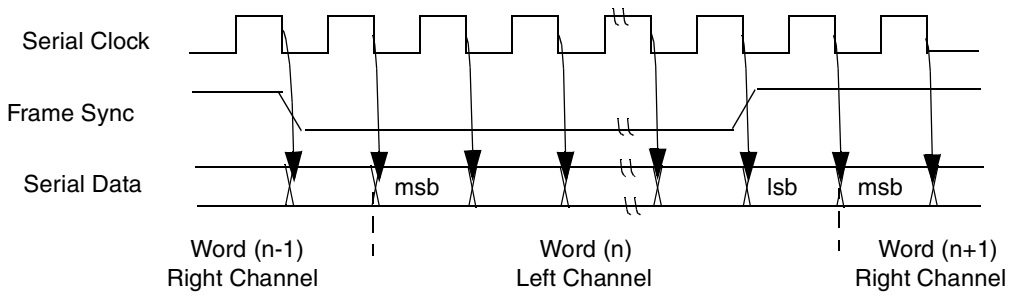


Figure 16-4. I2S Mode Timing—Serial Clock, Frame Sync and Serial Data

Select I^{2}S mode using the options listed in Table 16-2.

Table 16-2. I^{2}S Mode Selection

I^{2}S_MODE[1]	I2S_MODE[0]	Mode Type
0	0	Normal mode
0	1	I2S master mode
1	0	I2S slave mode
1	1	Normal mode

In normal mode operation, no register bits are forced to any particular state internally and the user can program the SSI to work in any operating condition.

When I2S modes are entered (I2S master (01) or I2S slave (10)), the following settings are recommended:

- Sync mode (SCR[4] =1)
- Tx shift direction: msb transmitted first (STCR[4]=0)
- Rx shift direction: msb received first (SRCR[4]=0)
- Tx data clocked at falling edge of the clock (STCR[3]=1)
- Rx data latched at rising edge of the clock (SRCR[3]=1)
- Tx frame sync active low (STCR[2]=1)
- Rx frame sync active low (SRCR[2]=1)
- Tx frame sync initiated one bit before data is transmitted (STCR[0]=1)
- Rx frame sync initiated one bit before data is received (SRCR[0]=1)

In I^{2}S master mode(SCR[6:5]=01), the following additional settings are recommended:

- TXDIR bit (STCR[5]) set to 1 to select internal generated bit clock
- TFDIR bit (STCR[6]) set to 1 to select internal generated frame sync

In I^{2}S master mode(SCR[6:5]=01), the following settings are internally overridden by the hardware:

- Network mode is selected (SCR[3]=1)
- Tx frame sync length set to one-word-long-frame (STCR[1]=0)

- Rx frame sync length set to one-word-long-frame (SRCR[1]=0)
- Tx shifting w.r.t. bit 0 of TXSR (STCR[9]=1)
- Rx shifting w.r.t. bit 0 of RXSR (SRCR[9]=1)

The user needs to set the following control bits to configure the bit clock and frame sync:

- PM (STCCR[7:0])
- PSR (STCCR[17])
- DIV2(STCCR[18]
- WL (STCCR[16:13]
- DC (STCCR[12:8])

The word length is fixed to 32 in I^2S Master mode and the WL bits determine the number of bits that will contain valid data (out of the 32 transmitted/received bits in each channel). The fixing of word duration as 32 simplifies the relation between oversampling clock (*ccm_ssi_clk*) and Frame Sync (*ccm_ssi_clk* becomes an integer multiple of Frame Sync).

In I^2S slave mode(SCR[6:5]=10), the following additional settings are recommended:

- TXDIR bit(STCR[5]) set to 0 to select external generated bit clock
- TFDIR bit(STCR[6]) set to 0 to select external generated frame sync

In I^2S slave mode(SCR[6:5]=10), the following settings are done internally overridden by the hardware:

- Normal mode is selected (SCR[3]=0)
- Tx frame sync length set to one-bit-long-frame (STCR[1]=1)
- Rx frame sync length set to one-bit-long-frame (SRCR[1]=1)
- Tx shifting w.r.t. bit 0 of TXSR (STCR[9]=1)
- Rx shifting w.r.t. bit 0 of RXSR (SRCR[9]=1)

The user needs to set the following control bits to configure the data transmission:

- WL (STCCR[16:13])
- DC (STCCR[12:8])

The word length is variable in I^2S slave mode and the WL bits determine the number of bits that will contain valid data. The actual word length is determined by the external CODEC. The external I^2S Master still sends frame sync according to the I^2S protocol (early, word wide and active low), the SSI internally operates so that each frame sync transition is the start of a new frame (the WL bits determine the number of bits to be transmitted/received). After one data word has been transferred, the SSI waits for the next frame sync transition to start operation in the next time slot. Transmit (STMSK) and receive (SRMSK) mask bits should not be used in I^2S slave mode of operation.

Note that when the SSI is operating in either I^2S slave or I^2S master mode, data is transmitted in both slots as per the I^2S standard, regardless of whether the SCR[3] bit is internally overridden to normal or network mode.

16.1.2.4 AC97 Mode

In AC97 mode of operation, the SSI transmits a 16-bit Tag Slot at the start of a frame and the rest of the slots (in that frame) are all 20-bits wide. The same sequence is followed while receiving data. Refer to the AC97 specification for details regarding transmit and receive sequences and data formats.

Note that the SSI only has one RxDATA pin so the SSI can only support one codec. Secondary codecs are not supported.

When AC97 mode is enabled, the following settings are internally overridden by the hardware. The programmed register values are not changed by entering AC97 mode but they no longer apply to the module's operation. Writing to the programmed register fields will update their values; these updates can be seen by reading back the register fields. However, these settings will not take effect until AC97 mode is turned off.

The register bits within the bracket are the equivalent settings:

- Sync mode is entered (SCR[4] =1)
- Network mode is selected (SCR[3]=1)
- Tx shift direction is msb transmitted first (STCR[4]=0)
- Rx shift direction is msb received first (SRCR[4]=0)
- Tx data is clocked at rising edge of the clock (STCR[3]=0)
- Rx data is latched at falling edge of the clock (SRCR[3]=0)
- Tx frame sync is active high (STCR[2]=0)
- Rx frame sync is active high (SRCR[2]=0)
- Tx frame sync length is one-word-long-frame (STCR[1]=0)
- Rx frame sync length is one-word-long-frame (SRCR[1]=0)
- Tx frame sync initiated one bit before data is transmitted (STCR[0]=1)
- Rx frame sync initiated one bit before data is received (SRCR[0]=1)
- Tx shifting w.r.t. bit 0 of TXSR (STCR[9]=1)
- Rx shifting w.r.t. bit 0 of RXSR (SRCR[9]=1)
- Tx FIFO is enabled (STCR[7]=1)
- Rx FIFO is enabled (SRCR[7]=1)
- TFDIR bit (STCR[6]) is forced to 1 internally to select internal generated frame sync
- TXDIR bit (STCR[5]) is forced to 0 internally to select external generated bit clock

Any alteration of these bits individually will not affect the operational conditions of the SSI unless AC97 mode is deselected.

Hence, the only control bits needed to be set by the user to configure the data transmission/reception are the WL (STCCR[16:13]) and DC (STCCR[12:8]) bits. In AC97 mode, the WL bits can only legally take the values corresponding to 16-bit (truncated data) or 20-bit time slots. In case WL bits are set to select 16-bit time slots, the SSI pads the transmit data (four least significant bits) with zeros and while receiving, stores only the most significant 16 bits in the Rx FIFO.

Follow the sequence for programming the SSI to work in AC97 mode:

1. Program the WL bits to a value corresponding to either 16 or 20 bits. The WL bit setting is only for the data portion of the AC97 frame (Slots #3 through #12). The Tag slot (Slot #0) is always 16 bits wide and the Command Address and Command Data slots (Slots #1 and #2) are always 20 bits wide.
2. Select the number of time slots by programming the DC bits. For AC97 operation, DC bits should be set to a value of '0xC', resulting in 13 time slots per frame.
3. Write data to be transmitted, in Tx FIFO 0 (through Tx Data Register 0)
4. Program the FV, TIF, RD, WR and FRDIV bits in SACNT register
5. Update the contents of SACADD, SACDAT and SATAG (for Fixed mode only) registers
6. Enable the AC97 mode of operation (AC97EN bit in SACNT register)

Once the SSI starts transmitting and receiving data (after being configured in AC97 mode), the programmer needs to service the interrupts, as and when they are raised (updates to command address/data or tag registers, reading of received data and writing more data for transmission). Further details regarding fixed and variable mode implementation are provided in the following sections.

While using AC97 in two-channel mode (TCH_EN=1), it is recommended that the received tag is not stored in the Rx FIFO (TIF=0). In case the programmer needs to update the SATAG register and also issue a RD/WR command (in a single frame), it is recommended that the SATAG register be updated prior to issuing a RD/WR command.

16.1.2.4.1 AC97 Fixed Mode (SACNT[1]=0)

In fixed mode of operation, SSI transmits in accordance with the Frame Rate Divider bits which decides the number of frames for which the SSI should be idle, after operating for one frame.

In a valid frame, TAG Value (written by Core) will be transmitted in Slot #0, Command Address will be transmitted in Slot #1 in case of RD/WR Command, and Command Data will be transmitted in Slot #2 in case of a WR Command. The data from TX-FIFO is transmitted in Slot #3 - Slot #12 depending on the valid slots indicated by the TAG value.

While receiving, bit 15 of the TAG Value (Slot #0) is checked to see if the CODEC is ready. If this bit is set, the frame is received. The received TAG provides the information about Slots containing valid data. The the corresponding TAG bit is valid, the Command Address (Slot #1) and Command Data (Slot #2) values are stored in the corresponding registers. The received data (Slot #3 - Slot #12) is then stored in the Rx-FIFO (for valid slots).

16.1.2.4.2 AC97 Variable Mode (SACNT[1]=1)

In Variable Mode, the transmit slots which should contain data in the current frame are determined by SLOTREQ bits received in the previous frame. While receiving, if the CODEC is ready, the frame is received and the SLOTREQ bits (contained in Slot #1) are stored for scheduling transmission in the next frame.

The SACCST, SACCEN and SACCDIS registers helps in determining which transmit slots are active. This information is used to ensure that SSI does not transmit data for powered-down/inactive channels.

16.1.2.5 External Frame and Clock Operation

When applying external frame sync and clock signals to SSI, there should be at least 4-bit clock cycles between the enabling of the transmit or receive section and the rising edge of the corresponding frame sync signal. The transition of STFS or SRFS should be synchronized with the rising edge of external clock signal, STCK or SRCK.

16.1.2.6 Data Alignment Formats Supported

The SSI supports three data formats in order to provide flexibility with handling data. These formats dictate how data is written to (and read from) the data registers. Therefore, data can appear in different places in STX0/1 and SRX0/1 based on the data format and the number of bits per word. Independent data formats are supported for both the transmitter and receiver (that is, the transmitter and receiver can use different data formats).

The supported data formats are:

- msb alignment
- lsb alignment
  - — Zero-extended (receive data only)
  - — Sign-extended (receive data only)

With msb alignment, the most significant byte is bits 31 through 24 of the data register if the word length is larger than or equal to 16 bits. If the word length is less than 16 bits and msb alignment is chosen, the most significant byte is bits 15 through 8. With lsb alignment, the least significant byte is bits 7 through 0. Data alignment is controlled by the TXBIT0 bit in the STCR and the RXBIT0 bit in the SRCR. See Table 16-3 for the bit assignment for all the data formats supported by the SSI.

Table 16-3. Data Alignment

Format	Bit Number																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
8-bit lsb Aligned																									7	6	5	4	3	2	1	0
8-bit msb Aligned																	7	6	5	4	3	2	1	0								
10-bit lsb Aligned																							9	8	7	6	5	4	3	2	1	0
10-bit msb Aligned																	9	8	7	6	5	4	3	2	1	0						
12-bit lsb Aligned																					11	10	9	8	7	6	5	4	3	2	1	0
12-bit msb Aligned																	11	10	9	8	7	6	5	4	3	2	1	0				
16-bit lsb Aligned																	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-bit msb Aligned	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																

Table 16-3. Data Alignment (continued)

Format	Bit Number																															
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18-bit lsb Aligned															17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
18-bit msb Aligned	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
20-bit lsb Aligned													19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
20-bit msb Aligned	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0												
22-bit lsb Aligned											21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
22-bit msb Aligned	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0										
24-bit lsb Aligned									23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
24-bit msb Aligned	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								

In addition, receive data can either be zero-extended or sign-extended if lsb alignment is selected. With zero-extension, all bits above the most significant bit are 0's. This format is useful when data is stored in a pure integer format. With sign-extension, all bits above the most significant bit are equal to the most significant bit. This format is useful when data is stored in a fixed-point integer format (which implies fractional values). Receive data extension is controlled by the RXEXT bit in the SRCR. Transmit data used with lsb alignment has no concept of sign/zero-extension. Unused bits above the most significant bit are simply ignored.

When configured in I2S or AC97 mode, the SSI forces the selection of lsb alignment. However, RXEXT still permits a choice between zero-extension and sign-extension.

Refer to Section 16.3.1.10, "SSI Transmit Configuration Register (STCR)," and Section 16.3.1.11, "SSI Receive Configuration Register (SRCR)," for more details on the relevant bits in the STCR and SRCR registers.

16.2 SSI External Signal Description

16.2.1 External Signals Overview

Table 16-4. Signal Properties

Name	Port	Function	Reset State	Pull up
SRCK	—	Serial Receive Clock	0	Passive
SRFS	—	Serial Receive Frame Sync	0	Passive
SRXD	—	Serial Receive Data	—	—
STCK	—	Serial Transmit Clock	0	Passive
STFS	—	Serial Transmit Frame Sync	0	Passive
STXD	—	Serial Transmit Data	0	Passive

16.2.2 SSI Detailed Signal Descriptions

16.2.2.1 SRCK—Serial Receive Clock

The SRCK port can be used as either an input or an output. This clock signal is used by the receiver and is always continuous.

16.2.2.2 SRFS—Serial Receive Frame Sync

The SRFS port can be used as either an input or an output. The frame sync is used by the receiver to synchronize the transfer of data. The frame sync signal can be one bit or one word in length and can occur one bit before the transfer of data or right at the transfer of data. If SRFS is configured as input, the external device should drive SRFS during rising edge of STCK or SRCK.

16.2.2.3 SRXD—Serial Receive Data

The SRXD port is an input and is used to bring serial data into the Receive Data Shift Register.

16.2.2.4 STCK—Serial Transmit Clock

The STCK port can be used as either an input or an output. This clock signal is used by the transmitter and is continuous. In Synchronous mode, this port is used by both the transmit and receive sections.

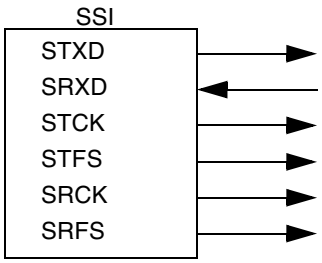
16.2.2.5 STFS—Serial Transmit Frame Sync

The STFS port can be used as either an input or an output. The frame sync is used by the transmitter to synchronize the transfer of data. The frame sync signal can be one bit or one word in length and can occur one bit before the transfer of data or right at the transfer of data. In Synchronous mode, this port is used by both the transmit and receive sections. If STFS is configured as input, the external device should drive STFS during rising edge of STCK or SRCK.

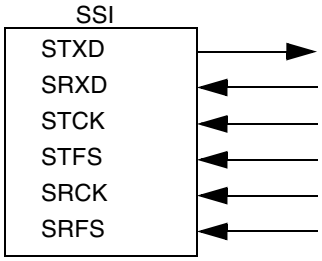
16.2.2.6 STXD—Serial Transmit Data

The STXD port is an output and transmits data from the Serial Transmit Shift Register. The STXD port is an output port when data is being transmitted and is disabled between data word transmissions and on the trailing edge of the bit clock after the last bit of a word is transmitted.

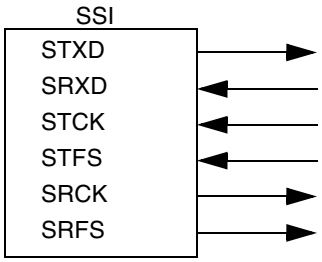
Figure 16-5 and Figure 16-6 show the main SSI configurations. These ports support all transmit and receive functions as shown.



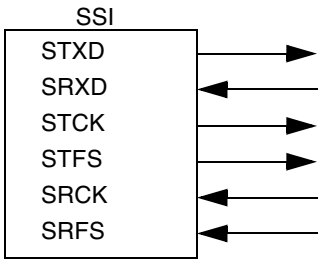
SSI Internal Continuous Clock for TX/RX (RXDIR=1,TXDIR=1,RFDIR=1,TFDIR=1,SYN=0)



SSI External Continuous Clock for TX/RX (RXDIR=0,TXDIR=0,RFDIR=0,TFDIR=0,SYN=0)

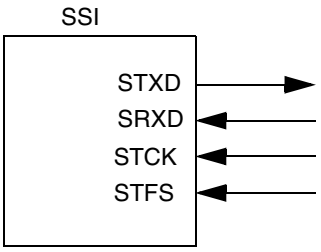


SSI Internal Continuous Clock for RX (RXDIR=1, TXDIR=0,RFDIR=1,TFDIR=0, SYN=0)
SSI External Continuous Clock for TX



SSI Internal Continuous Clock for TX (RXDIR=0, TXDIR=1, RFDIR=0, TFDIR=1, SYN=0)
SSI EXternal Continuous Clock for RX

Figure 16-5. Asynchronous (SYN=0) SSI Configurations—Continuous Clock



SSI External Continuous Clock (RXDIR=0, TXDIR=0, RFDIR=X, TFDIR=0,SYN=1)
SSI I2S Slave Mode(I2S_Mode=10)

Figure 16-6. Synchronous SSI Configuration—Continuous Clock

See Figure 16-7 for an example of the port signals for an 8-bit data transfer. Continuous clock signals are shown, as well as the bit-length frame sync signal and the word-length frame sync signal.

NOTE

The shift direction can be defined as msb first or lsb first and that there are other options on the clock and frame sync.

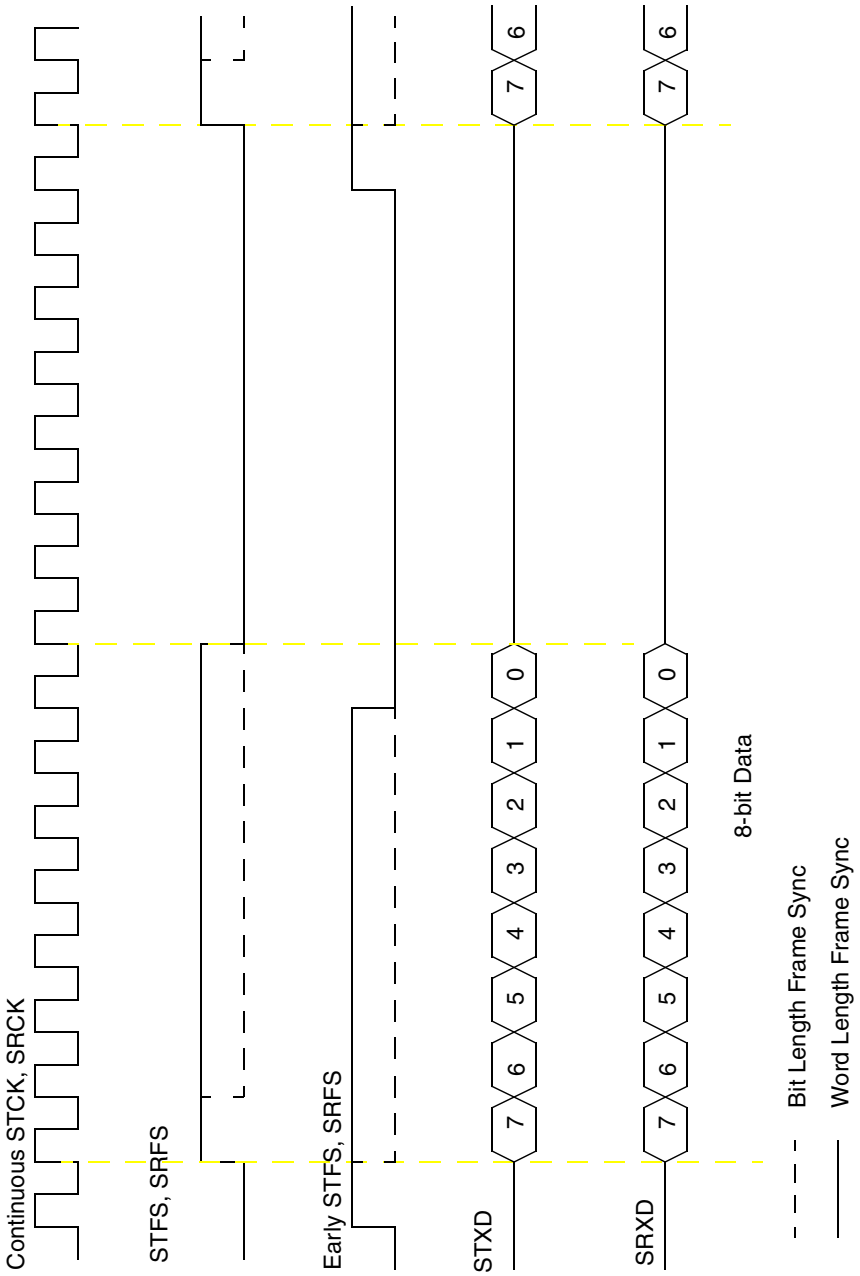


Figure 16-7. Serial Clock and Frame Sync Timing

See Table 16-5 for list of clock pin configurations.

Table 16-5. Clock Pin Configurations

SYN	RXDIR	TXDIR	RFDIR	TFDIR	SRCK	STCK	SRFS	STFS
Asynchronous Mode								
0	0	0	0	0	RCK in	TCK in	RFS in	TFS in
0	0	0	0	1	RCK in	TCK in	RFS in	TFS out

Table 16-5. Clock Pin Configurations (continued)

SYN	RXDIR	TXDIR	RFDIR	TFDIR	SRCK	STCK	SRFS	STFS
0	0	0	1	0	RCK in	TCK in	RFS out	TFS in
0	0	0	1	1	RCK in	TCK in	RFS out	TFS out
0	0	1	0	0	RCK in	TCK out	RFS in	TFS in
0	0	1	0	1	RCK in	TCK out	RFS in	TFS out
0	0	1	1	0	RCK in	TCK out	RFS out	TFS in
0	0	1	1	1	RCK in	TCK out	RFS out	TFS out
0	1	0	0	0	RCK out	TCK in	RFS in	TFS in
0	1	0	0	1	RCK out	TCK in	RFS in	TFS out
0	1	0	1	0	RCK out	TCK in	RFS out	TFS in
0	1	0	1	1	RCK out	TCK in	RFS out	TFS out
0	1	1	0	0	RCK out	TCK out	RFS in	TFS in
0	1	1	0	1	RCK out	TCK out	RFS in	TFS out
0	1	1	1	0	RCK out	TCK out	RFS out	TFS in
0	1	1	1	1	RCK out	TCK out	RFS out	TFS out
Synchronous Mode								
1	0	0	x	0	—	CK in	—	FS in
1	0	0	x	1	—	CK in	—	FS out
1	0	1	x	0	—	CK out	—	FS in
1	0	1	x	1	—	CK out	—	FS out

16.3 SSI Memory Map/Register Definition

The SSI registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR), plus the block base address, plus the offset of the specific register to be accessed. Table 16-6 lists the SSI-specific registers 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 CCSRBAR together with the block base address and offset listed in Table 16-6. The offsets to the memory map table are defined for both SSI interfaces. That is, SSI1 starts at address offset 0x000, and SSI2 starts at address offset 0x100. The registers for SSI1 are listed in Table 16-6, but the registers for SSI2 are not. Note that the registers are the same for SSI2 except that the offsets change from 0x0*nn* to 0x1*nn*.

NOTE

All SSI registers must be accessed as aligned 4-byte quantities. Accesses to the SSI registers that are less than 4-bytes are not supported.

Table 16-6. SSI Register Map

Offset	Register	Access	Reset	Section/Page
SSI1 Registers—Block Base Address 0x1_6000				
000	STX0—SSI Transmit Data Register 0	R/W	All zeros	16.3.1.1/16-25
004	STX1—SSI Transmit Data Register 1	R/W	All zeros	
008	SRX0—SSI Receive Data Register 0	Read-only	All zeros	16.3.1.4/16-28
00C	SRX1—SSI Receive Data Register 1	Read-only	All zeros	
010	SCR—SSI Control Register	R/W	All zeros	16.3.1.7/16-31
014	SISR—SSI Interrupt Status Register	Mixed	0x0000_3003	16.3.1.8/16-33
018	SIER—SSI Interrupt Enable Register	R/W	0x0000_3003	16.3.1.9/16-38
01C	STCR—SSI Transmit Configuration Register	R/W	0x0000_0200	16.3.1.10/16-39
020	SRCR—SSI Receive Configuration Register	R/W	0x0000_0200	16.3.1.11/16-41
024	STCCR—SSI Transmit Clock Control Register	R/W	0x0004_0000	16.3.1.12/16-42
028	SRCCR—SSI Receive Clock Control Register	R/W	0x0004_0000	
02C	SFCSR—SSI FIFO Control/Status Register	R/W	0x0081_0081	16.3.1.13/16-45
038	SACNT—SSI AC97 Control Register	R/W	All zeros	16.3.1.14/16-48
03C	SACADD—SSI AC97 Command Address Register	R/W	All zeros	16.3.1.15/16-49
040	SACDAT—SSI AC97 Command Data Register	R/W	All zeros	16.3.1.16/16-50
044	SATAG—SSI AC97 Tag Register	R/W	All zeros	16.3.1.17/16-50
048	STMSK—SSI Transmit Time Slot Mask Register	R/W	All zeros	16.3.1.18/16-51
04C	SRMSK—SSI Receive Time Slot Mask Register	R/W	All zeros	16.3.1.19/16-51
050	SACCST—SSI AC97 Channel Status Register	R	All zeros	16.3.1.20/16-52
054	SACCEN—SSI AC97 Channel Enable Register	W	All zeros	16.3.1.21/16-52
058	SACCDIS—SSI AC97 Channel Disable Register	W	All zeros	16.3.1.22/16-53
SSI2 Registers—Block Base Address 0x1_6000				
100–158	SSI2 Registers[1]	—	—	—

[1] SSI2 has the same memory-mapped registers that are described for SSI1 from 0x000 to 0x058, except the offsets range from 0x100 to 0x158.

Note that following registers are not accessible when the SSI is disabled (SCR[SSIEN] = 0):

- STX0
- SRX0
- STX1
- SRX1
- SACNT

- SACADD
- SACDAT
- SATAG
- STMSK
- SRMSK
- SACCST

16.3.1 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams in bit order.

16.3.1.1 SSI Transmit Data Registers 0 and 1 (STX0/1)

Figure 16-8 shows the SSI transmit data register 0 (STX0).

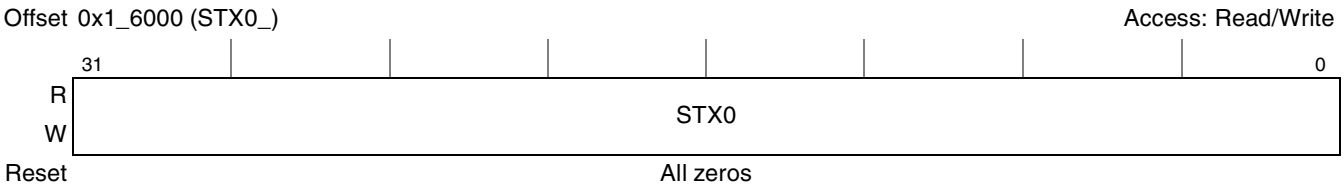


Figure 16-8. SSI0 Transmit Data Register 0 (STX0)

Figure 16-9 shows the SSI transmit data register 1 (STX1).

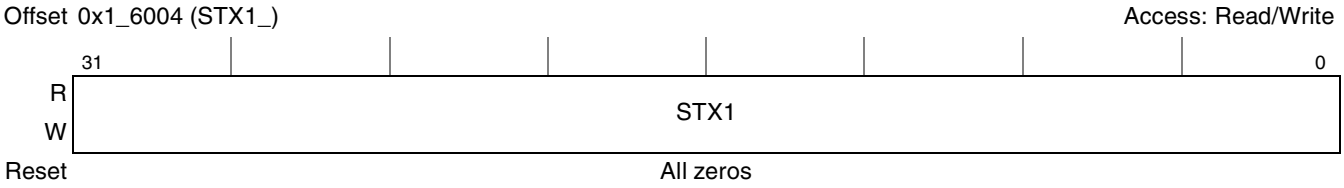


Figure 16-9. SSI1 Transmit Data Register 1 (STX1)

Table 16-7 describes the STX*n* fields.

Table 16-7. STX*n* Field Descriptions

Bits	Name	Description
31–0	STX*n*	SSI Transmit Data. These bits store the data to be transmitted by the SSI. These are implemented as the first word of their respective Tx FIFOs. Data written to these registers is transferred to the Transmit Shift Register (TXSR), when shifting of the previous data is complete. If both FIFOs are in use, data is alternately transferred from STX0 and STX1, to TXSR. Multiple writes to the STX registers will not result in the previous data being over-written by the subsequent data. STX1 can only be used in two-channel mode of operation. Protection from over-writing is present irrespective of whether the transmitter is enabled or not. Example 1: If Tx FIFO0 is in use and user writes Data1...Data9 to STX0, Data9 will not over-write Data1. Data1...Data8 are stored in the FIFO while Data9 is discarded. Example 2: If Tx FIFO0 is not in use and user writes Data1, Data2 to STX0, then Data2 will not over-write Data1 and will be discarded.

NOTE

Enable SSI (SSIEN = 1) before writing to the SSI transmit data registers.

16.3.1.2 SSI Transmit FIFO 0 and 1 Registers

The SSI transmit FIFO registers are 8x32-bit registers. These registers are not directly accessible by the end user (except in SSI test mode). Transmit Shift Register (TXSR) receives its values from these FIFO registers. Transmitted data is first-in-first-out. When the Transmit Interrupt Enable (TIE) bit in the SIER and either of the Transmit FIFO Empty (TFE0 or 1) bits in the SISR are set, the core is interrupted whenever the data level in either of the SSI Transmit FIFOs falls below the selected threshold.

16.3.1.3 SSI Transmit Shift Register (TXSR)

The SSI Transmit Shift Register (TXSR) is a 24-bit shift register that contains the data being transmitted. This register is not directly accessible by the end user. When a continuous clock is used, data is shifted out to the Serial Transmit Data (STXD) port by the selected (internal/external) bit clock when the associated (internal/external) frame sync is asserted. The Word Length control bits (WL[3:0]) in the STCCR, described in Section 16.3.1.12, “SSI Transmit and Receive Clock Control Registers (STCCR and SRCCR),” determine the number of bits to be shifted out of the TXSR before it is considered empty and can be written to again. The word length can be 8, 10, 12, 16, 18, 20, 22 or 24 bits. The data to be transmitted occupies the most significant portion of the shift register if TXBIT0 is cleared; otherwise, it occupies the least significant portion. The unused portion of the register is ignored.

Data is shifted out of this register with the most significant bit (msb) of the data word first when STCR[TSHFD] is cleared. Note that when TXBIT0 is cleared, for word lengths less than 16 bits, the data is shifted out of bit 15 of the transmit data register (STX). If STCR[TSHFD] is set, the least significant bit (lsb) of the data word is shifted out first.

The following figures show the transmit data path for the four configurations of alignment and shifting using representative word length values; operation is similar for word length values not shown.

Figure 16-10 shows the transmitter loading and shifting operation for the msb-aligned and msb-first (STCR[TXBIT, TSHFD] = 0,0) case.

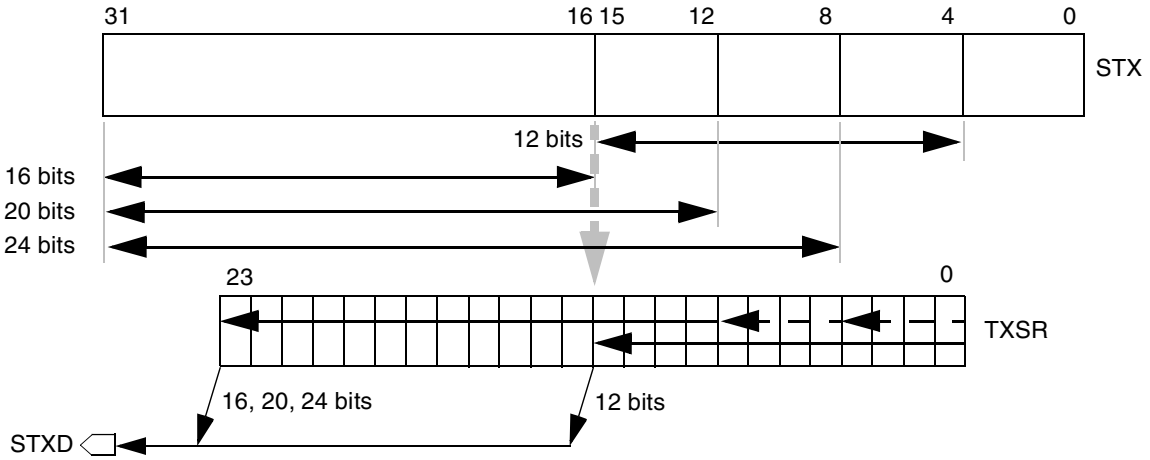


Figure 16-10. Transmit Data Path—msb-Aligned, msb-First (TXBIT0 = 0, TSHFD = 0)

Figure 16-11 shows the transmitter loading and shifting operation for the msb-aligned and lsb-first (STCR[TXBIT, TSHFD] = 0,1) case.

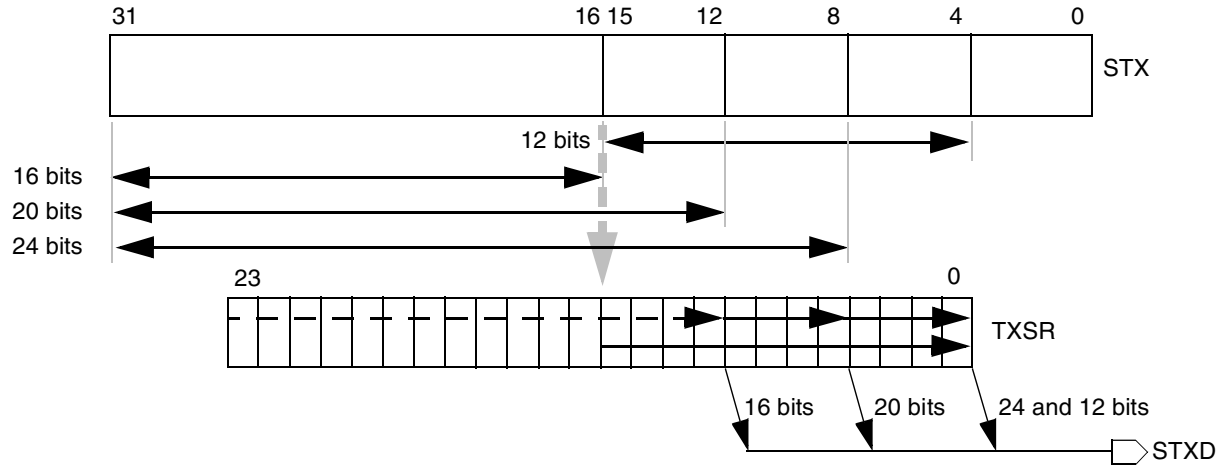


Figure 16-11. Transmit Data Path—msb-Aligned, lsb-First (TXBIT0 = 0, TSHFD = 1)

Figure 16-12 shows the transmitter loading and shifting operation for the lsb-aligned and msb-first (STCR[TXBIT, TSHFD] = 1,0) case.

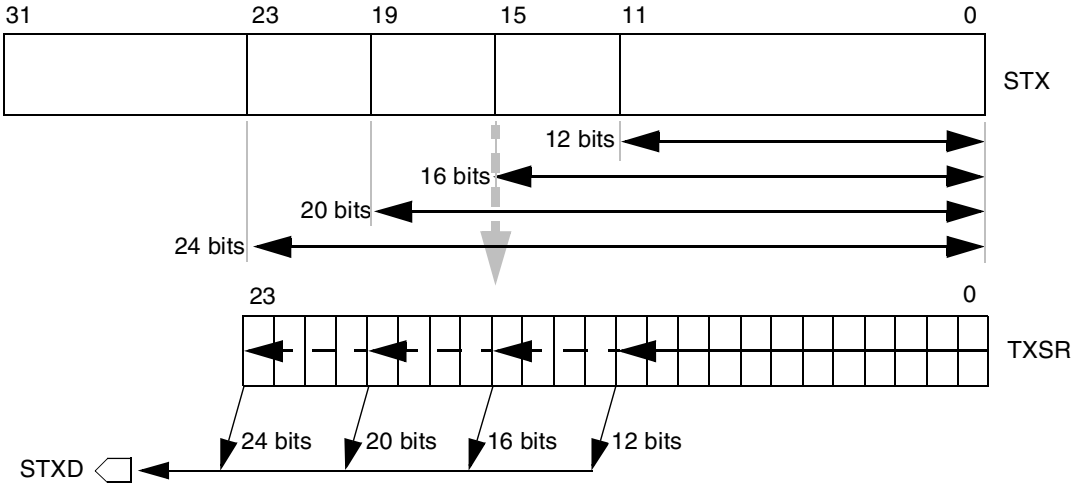


Figure 16-12. Transmit Data Path—lsb-Aligned, msb-First (TXBIT0=1, TSHFD=0)

Figure 16-13 shows the transmitter loading and shifting operation for the lsb-aligned and lsb-first (STCR[TXBIT, TSHFD] = 1,1) case.

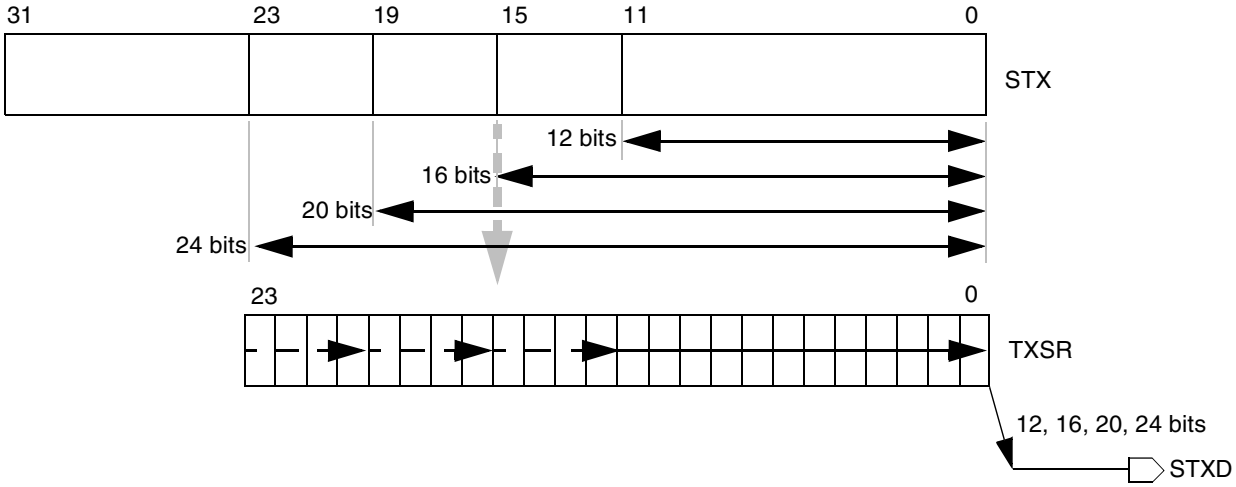


Figure 16-13. Transmit Data Path—lsb-Aligned, lsb-First (TXBIT0=1, TSHFD=1)

16.3.1.4 SSI Receive Data Registers 0 and 1 (SRX0/1)

Figure 16-14 shows the SSI receive data register 0.

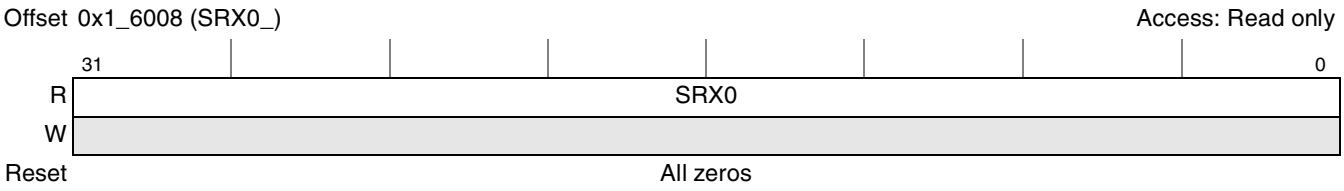


Figure 16-14. SSI0 Receive Data Register 0 (SRX0)

Figure 16-15 shows the SSI receive data register 1.

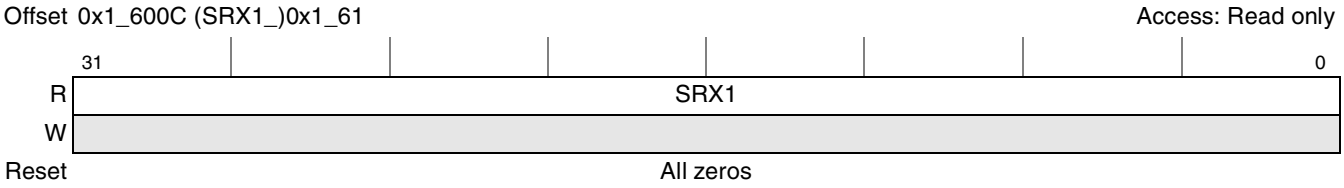


Figure 16-15. SSI0 Receive Data Register 0 (SRX1)

Table 16-8 describes the SRX*n* fields.

Table 16-8. SRX*n* Field Descriptions

Bits	Name	Description
31–0	SRX*n*	SSI Receive Data. These bits store the data received by the SSI. These are implemented as the first word of their respective Rx FIFOs. These bits receive data from the RXSR depending on the mode of operation. In case both FIFOs are in use, data is transferred to each data register alternately. SRX1 can only be used in two-channel mode of operation.

16.3.1.5 SSI Receive FIFO 0 and 1 Registers

The SSI Receive FIFO Registers are 8x32-bit registers. These registers are not directly accessible by the end user (except in SSI test mode). They always accepts data from the Receive Shift Register (RXSR). The core is interrupted when the data level in either of the SSI Receive FIFOs reaches the selected threshold, if the associated interrupt is enabled.

16.3.1.6 SSI Receive Shift Register (RXSR)

The SSI Receive Shift Register (RXSR) is a 24-bit, shift register that receives incoming data from the serial receive data SRXD port. This register is not directly accessible by the end user. When a continuous clock is used, data is shifted in by the selected (internal/external) bit clock when the associated (internal/external) frame sync is asserted.

Data is transferred to the appropriate SSI Receive Data Register (SRX0/1) or Receive FIFOs (if the receive FIFO is enabled and the corresponding SRX is full) after 8, 10, 12, 16, 18, 20, 22 or 24 bits have been shifted in depending on the word length as specified by SRCCR[WL]. The data received occupies the most significant portion of the shift register if SRCR[RXBIT0] is cleared; otherwise data occupies the least significant portion. When receiving word lengths less than 24 bits, the unused bits are filled with zeros. Note that when SRCR[RXBIT0] is cleared, for word lengths less than 16 bits, the data is shifted into bit 15 of the FIFO or receive data register (SRX). When SRCR[RSHFD] is cleared. data is received msb first; when SRCR[RSHFD] is set, data is received lsb first.

The following figures show the receiver data path for the four configurations of alignment and shifting using representative word lengths; operation is similar for word lengths not shown. Figure 16-16 shows

the receiver data loading and shifting operation for the msb-aligned, msb-first (SRCR[RXBIT0, RSHFD] = 0,0) case.

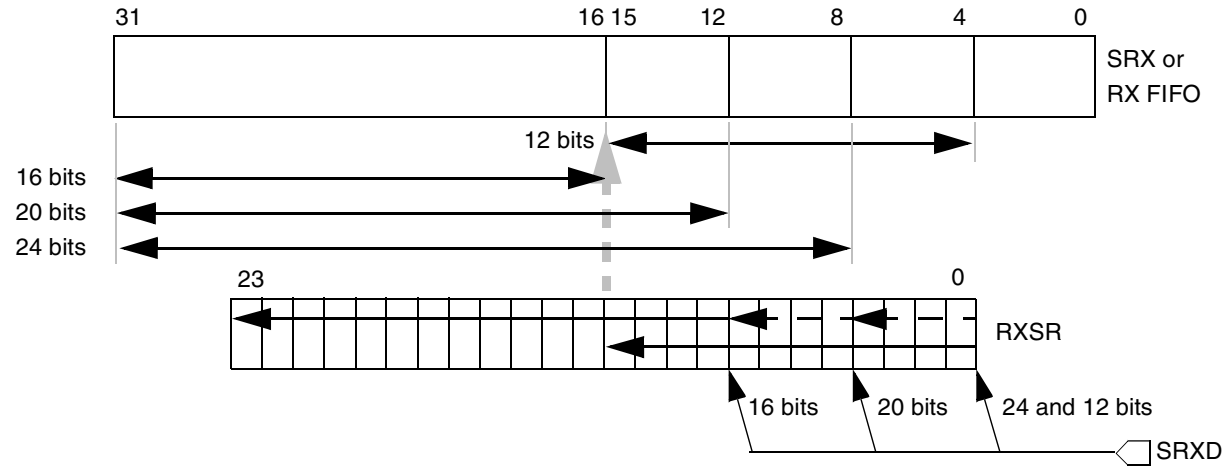


Figure 16-16. Receive Data Path—msb-Aligned, msb-First (RXBIT0=0, RSHFD=0)

Figure 16-17 shows the receiver data loading and shifting operation for the msb-aligned, msb-first (SRCR[RXBIT0, RSHFD] = 0,0) case.

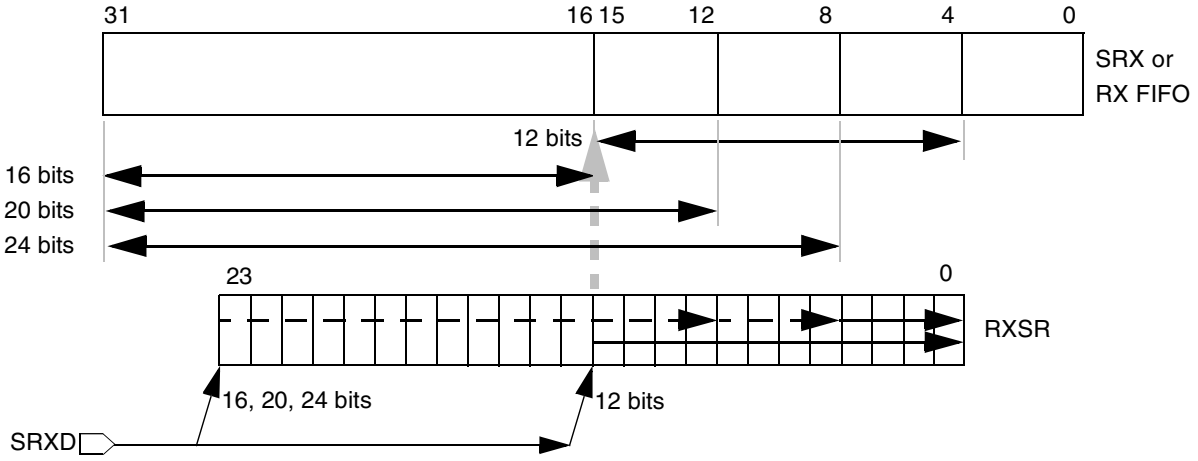


Figure 16-17. Receive Data Path—msb-Aligned, lsb-First (RXBIT0=0, RSHFD=1)

Figure 16-18 shows the receiver data loading and shifting operation for the msb-aligned, msb-first (SRCR[RXBIT0, RSHFD] = 0,0) case.

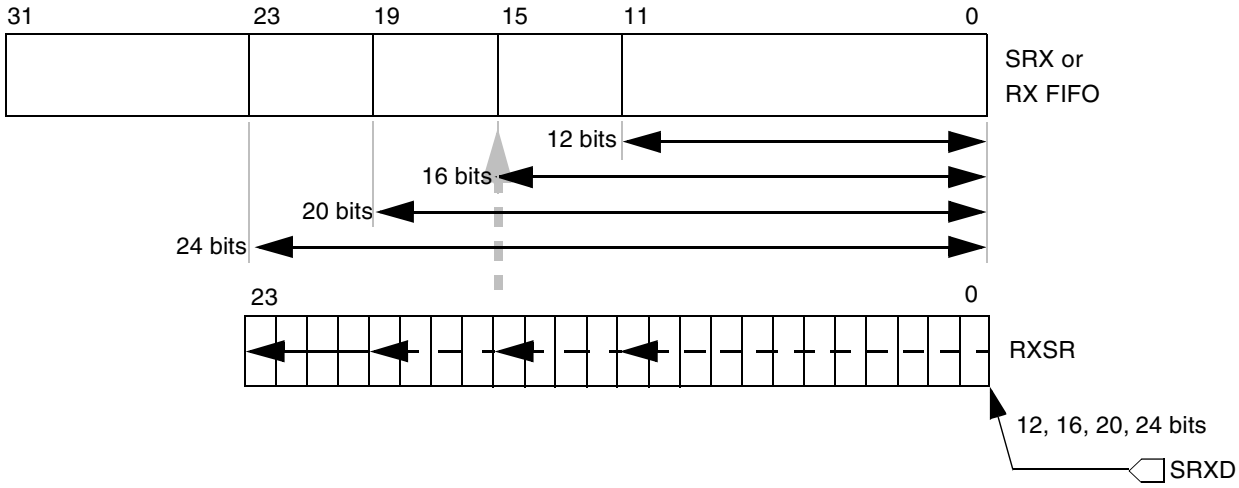


Figure 16-18. Receive Data Path—lsb-Aligned, msb-First (RXBIT0=1, RSHFD=0)

Figure 16-19 shows the receiver data loading and shifting operation for the msb-aligned, msb-first (SRCR[RXBIT0, RSHFD] = 0,0) case.

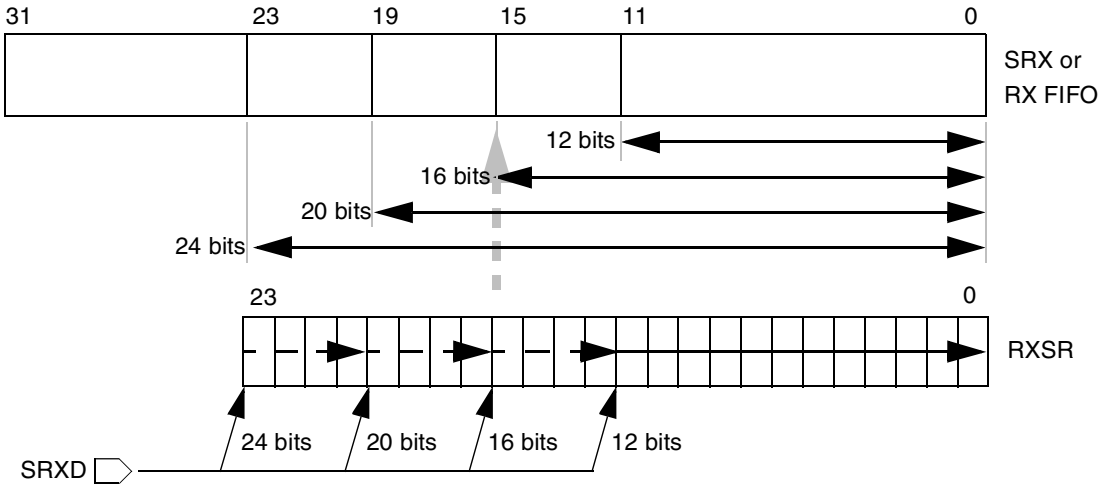


Figure 16-19. Receive Data Path—lsb-Aligned, lsb-First (RXBIT0=1, RSHFD=1)

16.3.1.7 SSI Control Register (SCR)

The SSI Control Register (SCR) is a 10-bit register used to set up the SSI. SSI reset is controlled by bit 0 in the SCR. SSI operating modes are also selected in this register (except AC97 mode which is selected in SACNT register).

Figure 16-20 shows the SSI control register (SCR).

Offset 0x1_6010 (SCR) 0x1_61 Access: Read/Write

	31–12	11	10	9	8	7	6	5	4	3	2	1	0
R W	—	RFR_CLK_DIS	TFR_CLK_DIS	—	TCH_EN	—	I2S MODE		SYN	NET	RE	TE	SSIEN
Reset	All zeros												

Figure 16-20. SSI Control Register (SCR)

Table 16-9 describes teh SCR fields.

Table 16-9. SCR Field Descriptions

Bits	Name	Description
31–12	—	Reserved
11	RFR_CLK_DIS	Receive Frame Clock Disable. This bit provide option to keep the Frame-sync and Clock enabled or disabled after current receive frame, in which receiver is disabled by clearing RE bit. Writing to this bit has effect only when RE is disabled. 0 Continue Frame-sync/Clock generation after current frame during which RE is cleared. This may be required when Frame-sync and Clocks are required from SSI, even when no data is to be received. 1 Stop Frame-sync/Clock generation at next frame boundary. This will be effective also in case where receiver is already disabled in current or previous frames.
10	TFR_CLK_DIS	Transmit Frame Clock Disable. This bit provide option to keep the Frame-sync and Clock enabled or disabled after current transmit frame, in which transmitter is disabled by clearing TE bit. Writing to this bit has effect only when TE is disabled. 0 Continue Frame-sync/Clock generation after current frame during which TE is cleared. This may be required when Frame-sync and Clocks are required from SSI, even when no data is to be received. 1 Stop Frame-sync/Clock generation at next frame boundary. This will be effective also in case where transmitter is already disabled in current or previous frames.
9	—	Reserved
8	TCH_EN	Two-channel operation enable. This bit allows SSI to operate in the two-channel mode. In this mode, 2 time slots should be used out of the possible 32. Any 2 time slots (from 0 to 31) can be selected. The data in the two time slots is alternately handled by the two data registers (0 and 1). While receiving, the RXSR transfers data to SRX0 and SRX1 alternately and while transmitting, data is alternately transferred from STX0 and STX1 to TXSR. If more than 2 slots are to be enabled, then for odd number of slots two-channel operation is deprecated and TCH_EN should be cleared. This will ensure that all data is received and transmitted from one fifo, FIFO0. For an even number of slots, two-channel operation can be enabled to optimize usage of both FIFOs or disabled as in the case of odd number of active slots. **Note:** The second channel does not have any DMA requests associated with it. Therefore if data transfer by DMA is desired, it must be ensured that the two channels remain in sync and the DMA controller must be configured such that each SSI DMA request reads or writes both channels (STX0 and STX1, or SRX0 and SRX1) as appropriate. 0 Two-channel mode disabled 1 Two-channel mode enabled
7	—	Reserved

Table 16-9. SCR Field Descriptions (continued)

Bits	Name	Description
6–5	I2S MODE	I2S Mode Select. These bits allow the SSI to operate in Normal, I2S Master or I2S Slave mode. See Section 16.1.2.3, "I2S Mode," for a detailed description of I2S Mode of operation. See Table 16-2 for details regarding settings.
4	SYN	Synchronous Mode. This bit controls whether SSI is in synchronous mode or not. In synchronous mode, the transmit and receive sections of SSI share a common clock port (STCK) and frame sync port (STFS). 0 Asynchronous mode selected 1 Synchronous mode selected
3	NET	Network Mode. This bit controls whether SSI is in network mode or not. 0 Network mode not selected 1 Network mode selected
2	RE	Receive Enable. This control bit enables the receive section of the SSI. When this bit is enabled, data reception starts with the arrival of the next frame sync. If data is being received when this bit is cleared, data reception continues until the end of the current frame and then stops. If this bit is set again before the second to last bit of the last time slot in the current frame, then reception continues without interruption. 0 Receive section disabled 1 Receive section enabled
1	TE	Transmit Enable. This control bit enables the transmit section of the SSI. It enables the transfer of the contents of the STX registers to the TXSR and also enables the internal transmit clock. The transmit section is enabled when this bit is set and a frame boundary is detected. When this bit is cleared, the transmitter continues to send data until the end of the current frame and then stops. Data can be written to the STX registers with the TE bit cleared (the corresponding TDE bit will be cleared). If the TE bit is cleared and then set again before the second to last bit of the last time slot in the current frame, data transmission continues without interruption. The normal transmit enable sequence is to write data to the STX register(s) and then set the TE bit. The normal disable sequence is to clear the TE and TIE bits after the TDE bit is set. 0 Transmit section disabled 1 Transmit section enabled
0	SSIEN	SSI Enable This bit is used to enable/disable the SSI. When disabled, all SSI status bits are preset to the same state produced by the power-on reset, all control bits are unaffected, the contents of Tx and Rx FIFOs are cleared. When SSI is disabled, all internal clocks are disabled (except register access clock). 0 SSI is disabled. 1 SSI is enabled.

16.3.1.8 SSI Interrupt Status Register (SISR)

The SSI Interrupt Status Register (SISR) is used to monitor the SSI. This register is used by the core to interrogate the status of the SSI. The status bits are described in the following table. Figure 16-21 shows a list of SSI interrupt sources. All Receiver related interrupts are generated only if the Receiver is enabled (SCR[2]=1) and all Transmitter related interrupts are generated only if the Transmitter is enabled (SCR[1]=1).

SSI Interrupt Sources
SSI Receive Data with Exception Status 0/1
SSI Receive Data 0/1
SSI Receive Last Time Slot
SSI Transmit Data with Exception Status 0/1
SSI Transmit Data 0/1
SSI Transmit Last Time Slot
SSI AC97 Command Address Updated
SSI AC97 Command Data Updated
SSI AC97 Receive Tag Updated
SSI Receive Frame Sync
SSI Transmit Frame Sync
SSI Receive Frame Complete
SSI Transmit Frame Complete

Figure 16-21. SSI Interrupts

NOTE

- SSI Status flags are valid when SSI is enabled.
- See Section 16.4.2, “Receive Interrupt Enable Bit Description,” and Section 16.4.3, “Transmit Interrupt Enable Bit Description,” for interrupt source mapping.
- All the flags in the SISR are updated after the first bit of the next SSI word has completed transmission or reception. Certain status bits (ROE0/1 and TUE0/1) are cleared by writing 1 to the corresponding interrupt status bit in SISR.

Figure 16-22 shows the SSI interrupt status register (SISR).

Offset 0x1_6014 (SISR) Access: Mixed

	31–25	24	23	22–19	18	17	16
R	—	RFRC	TRFC	—	CMDAU	CMDDU	RXT
W		w1c	w1c				
Reset	All zeros						

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RDR1	RDR0	TDE1	TDE0	ROE1	ROE0	TUE1	TUE0	TFS	RFS	TLS	RLS	RFF1	RFF0	TFE1	TFE0
W					w1c	w1c	w1c	w1c								
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Figure 16-22. SSI Interrupt Status Register (SISR)

Table 16-10 describes the SISR fields.

Table 16-10. SISR Field Descriptions

Bits	Name	Description
31–25	—	Reserved
24	RFRC	Receive Frame Complete. This flag is set at the end of the frame during which Receiver is disabled. If Receive Frame & Clock are not disabled in the same frame, this flag is also set at the end of the frame in which Receive Frame & Clock are disabled. See description of RFR_CLK_DIS bit in Section 16.3.1.7, "SSI Control Register (SCR)," for more details on how to disable Receiver Frame & Clock or keep them enabled after receiver is disabled. 0 End of Frame not reached 1 End of frame reached after disabling RE or disabling RFR_CLK_DIS, when receiver is already disabled.
23	TFRC	Transmit Frame Complete. This flag is set at the end of the frame during which Transmitter is disabled. If Transmit Frame & Clock are not disabled in the same frame, this flag is also set at the end of the frame in which Transmit Frame & Clock are disabled. See description of TFR_CLK_DIS bit in Section 16.3.1.7, "SSI Control Register (SCR)," for more details on how to disable Receiver Frame & Clock or keep them enabled after receiver is disabled. 0 End of Frame not reached 1 End of frame reached after disabling TE or disablingTFR_CLK_DIS, when transmitter is already disabled.
22–19	—	Reserved
18	CMDAU	Command Address Register Updated. This bit causes the Command Address Updated interrupt (when CMDAU_EN bit is set). This status bit is set each time there is a difference in the previous and current value of the received Command Address. This bit is cleared on reading the SACADD register. 0 No change in SACADD register. 1 SACADD register updated with different value.
17	CMDDU	Command Data Register Updated. This bit causes the Command Data Updated interrupt (when CMDDU_EN bit is set). This status bit is set each time there is a difference in the previous and current value of the received Command Data. This bit is cleared on reading the SACDAT register. 0 No change in SACDAT register. 1 SACDAT register updated with different value.
16	RXT	Receive Tag Updated. This status bit is set each time there is a difference in the previous and current value of the received tag. It causes the Receive Tag Interrupt (if RXT_EN bit is set). This bit is cleared on reading the SATAG register. 0 No change in SATAG register. 1 SATAG register updated with different value.
15	RDR1	Receive Data Ready 1. This flag bit is set when SRX1 or Rx FIFO 1 is loaded with a new value and two-channel mode is selected. RDR1 is cleared when the Core reads the SRX1 register. If Rx FIFO 1 is enabled, RDR1 is cleared when the FIFO is empty. If RIE and RDR1_EN are set, a Receive Data 1 interrupt request is issued on setting of RDR1 bit in case Rx FIFO1 is disabled, if the FIFO is enabled, the interrupt is issued on RFF1 assertion. The RDR1 bit is cleared by POR and SSI reset. 0 No new data for Core to read. 1 New data for Core to read.
14	RDR0	Receive Data Ready 0. This flag bit is set when SRX0 or Rx FIFO 0 is loaded with a new value. RDR0 is cleared when the Core reads the SRX0 register. If Rx FIFO 0 is enabled, RDR0 is cleared when the FIFO is empty. If RIE and RDR0_EN are set, a Receive Data 0 interrupt request is issued on setting of RDR0 bit in case Rx FIFO0 is disabled, if the FIFO is enabled, the interrupt is issued on RFF0 assertion. The RDR0 bit is cleared by POR and SSI reset. 0 No new data for Core to read. 1 New data for Core to read.

Table 16-10. SISR Field Descriptions (continued)

Bits	Name	Description
13	TDE1	Transmit Data Register Empty 1. This flag is set whenever data is transferred to TXSR from STX1 register and two-channel mode is selected. If Tx FIFO1 is enabled, this occurs when there is at least one empty slot in STX1 or Tx FIFO1. If Tx FIFO1 is not enabled, this occurs when the contents of STX1 are transferred to TXSR. The TDE1 bit is cleared when the Core writes to STX1. If TIE and TDE1_EN are set, an SSI Transmit Data 1 interrupt request is issued on setting of TDE1 bit. The TDE1 bit is cleared by POR and SSI reset. 0 Data available for transmission. 1 Data needs to be written by the Core for transmission.
12	TDE0	Transmit Data Register Empty 0. This flag is set whenever data is transferred to TXSR from STX0 register. If Tx FIFO 0 is enabled, this occurs when there is at least one empty slot in STX0 or Tx FIFO 0. If Tx FIFO 0 is not enabled, this occurs when the contents of STX0 are transferred to TXSR. The TDE0 bit is cleared when the Core writes to STX0. If TIE and TDE0_EN are set, an SSI Transmit Data 0 interrupt request is issued on setting of TDE0 bit. The TDE0 bit is cleared by POR and SSI reset. 0 Data available for transmission. 1 Data needs to be written by the Core for transmission.
11	ROE1	Receiver Overrun Error 1. This flag is set when the RXSR is filled and ready to transfer to SRX1 register or to Rx FIFO 1 (when enabled) and these are already full and two-channel mode is selected. If Rx FIFO 1 is enabled, this is indicated by RFF1 flag, else this is indicated by the RDR1 flag. The RXSR is not transferred in this case. The ROE1 flag causes an interrupt if RIE and ROE1_EN are set. The ROE1 bit is cleared by POR and SSI reset. It is also cleared by writing ‘1’ to this bit. Clearing the RE bit does not affect the ROE1 bit. 0 Default interrupt issued to the Core. 1 Exception interrupt issued to the Core.
10	ROE0	Receiver Overrun Error 0. This flag is set when the RXSR is filled and ready to transfer to SRX0 register or to Rx FIFO 0 (when enabled) and these are already full. If Rx FIFO 0 is enabled, this is indicated by RFF0 flag, else this is indicated by the RDR0 flag. The RXSR is not transferred in this case. The ROE0 flag causes an interrupt if RIE and ROE0_EN are set. The ROE0 bit is cleared by POR and SSI reset. It is also cleared by writing ‘1’ to this bit. Clearing the RE bit does not affect the ROE0 bit. 0 Default interrupt issued to the Core. 1 Exception interrupt issued to the Core.
9	TUE1	Transmitter Underrun Error 1. This flag is set when the TXSR is empty (no data to be transmitted), the TDE1 flag is set, a transmit time slot occurs and the SSI is in two-channel mode. When a transmit underrun error occurs, the previous data is retransmitted. In Network mode, each time slot requires data transmission (unless masked through STMSK register), when the transmitter is enabled (TE is set). The TUE1 flag causes an interrupt if TIE and TUE1_EN are set. The TUE1 bit is cleared by POR and SSI reset. It is also cleared by writing ‘1’ to this bit. 0 Default interrupt issued to the Core. 1 Exception interrupt issued to the Core.
8	TUE0	Transmitter Underrun Error 0. This flag is set when the TXSR is empty (no data to be transmitted), the TDE0 flag is set and a transmit time slot occurs. When a transmit underrun error occurs, the previous data is retransmitted. In Network mode, each time slot requires data transmission (unless masked through STMSK register), when the transmitter is enabled (TE is set). The TUE0 flag causes an interrupt if TIE and TUE0_EN are set. The TUE0 bit is cleared by POR and SSI reset. It is also cleared by writing ‘1’ to this bit. 0 Default interrupt issued to the Core. 1 Exception interrupt issued to the Core.

Table 16-10. SISR Field Descriptions (continued)

Bits	Name	Description
7	TFS	Transmit Frame Sync. This flag indicates the occurrence of transmit frame sync. Data written to the STX registers during the time slot when the TFS flag is set, is sent during the second time slot (in Network mode) or in the next first time slot (in Normal mode). In Network mode, the TFS bit is set during transmission of the first time slot of the frame and is then cleared when starting transmission of the next time slot. In Normal mode, this bit is always high. This flag causes an interrupt if TIE and TFS_EN are set. The TFS bit is cleared by POR and SSI reset. 0 No Occurrence of Transmit frame sync. 1 Transmit frame sync occurred during transmission of last word written to STX registers.
6	RFS	Receive Frame Sync. This flag indicates the occurrence of receive frame sync. In Network mode, the RFS bit is set when the first slot of the frame is being received. It is cleared when the next slot begins to be received. In Normal mode, this bit is always high. This flag causes an interrupt if RIE and RFS_EN are set. The RFS bit is cleared by POR and SSI reset. 0 No Occurrence of Receive frame sync. 1 Receive frame sync occurred during reception of next word in SRX registers.
5	TLS	Transmit Last Time Slot. This flag indicates the last time slot in a frame. When set, it indicates that the current time slot is the last time slot of the frame. TLS is set at the start of the last transmit time slot and causes the SSI to issue an interrupt (if TIE and TLS_EN are set). TLS is cleared when the SISR is read with this bit set. The TLS bit is cleared by POR and SSI reset. 0 Current time slot is not last time slot of frame. 1 Current time slot is the last transmit time slot of frame.
4	RLS	Receive Last Time Slot. This flag indicates the last time slot in a frame. When set, it indicates that the current time slot is the last receive time slot of the frame. RLS is set at the end of the last time slot and causes the SSI to issue an interrupt (if RIE and RLS_EN are set). RLS is cleared when the SISR is read with this bit set. The RLS bit is cleared by POR and SSI reset. 0 Current time slot is not last time slot of frame. 1 Current time slot is the last receive time slot of frame.
3	RFF1	Receive FIFO Full 1. This flag is set when Rx FIFO1 is enabled, the data level in Rx FIFO1 reaches the selected Rx FIFO WaterMark 1 (RFWM1) threshold and the SSI is in two-channel mode. The setting of RFF1 only causes an interrupt when RIE and RFF1_EN are set, Rx FIFO1 is enabled and the two-channel mode is selected. RFF1 is automatically cleared when the amount of data in Rx FIFO1 falls below the threshold. The RFF1 bit is cleared by POR and SSI reset. When Rx FIFO1 contains 8 words, the maximum it can hold, all further data received (for storage in this FIFO) is ignored until the FIFO contents are read. 0 Space available in Receive FIFO1. 1 Receive FIFO1 is full.
2	RFF0	Receive FIFO Full 0. This flag is set when Rx FIFO0 is enabled and the data level in Rx FIFO0 reaches the selected Rx FIFO WaterMark 0 (RFWM0) threshold. The setting of RFF0 only causes an interrupt when RIE and RFF0_EN are set and Rx FIFO0 is enabled. RFF0 is automatically cleared when the amount of data in Rx FIFO0 falls below the threshold. The RFF0 bit is cleared by POR and SSI reset. When Rx FIFO0 contains 8 words, the maximum it can hold, all further data received (for storage in this FIFO) is ignored until the FIFO contents are read. 0 Space available in Receive FIFO0. 1 Receive FIFO0 is full.

Table 16-10. SISR Field Descriptions (continued)

Bits	Name	Description
1	TFE1	Transmit FIFO Empty 1. This flag is set when Tx FIFO1 is enabled, the data level in Tx FIFO1 falls below the selected Tx FIFO WaterMark 1 (TFWM1) threshold and the two-channel mode is selected. The setting of TFE1 only causes an interrupt when TIE and TFE1_EN are set, Tx FIFO1 is enabled and two-channel mode is selected. The TFE1 bit is automatically cleared when the data level in Tx FIFO1 becomes more than the amount specified by the watermark bits. The TFE1 bit is set by POR and SSI reset. 0 Transmit FIFO1 has data for transmission. 1 Transmit FIFO1 is empty.
0	TFE0	Transmit FIFO Empty 0. This flag is set when Tx FIFO0 is enabled and the data level in Tx FIFO0 falls below the selected Tx FIFO WaterMark 0 (TFWM0) threshold. The setting of TFE0 only causes an interrupt when TIE and TFE0_EN are set and Tx FIFO0 is enabled. The TFE0 bit is automatically cleared when the data level in Tx FIFO0 becomes more than the amount specified by the watermark bits. The TFE0 bit is set by POR and SSI reset. 0 Transmit FIFO0 has data for transmission. 1 Transmit FIFO0 is empty.

16.3.1.9 SSI Interrupt Enable Register (SIER)

The SSI interrupt enable register (SIER), shown in Figure 16-23, is a 25-bit register used to set up the SSI interrupts and DMA requests.

Offset 0x1_6018 (SIER) Access: User read/write

	31–25	24	23	22	21	20	19	18	17	16
R W	—	RFRC_EN	TFRC_EN	RDMAE	RIE	TDMAE	TIE	CMDAU_EN	CMDDU_EN	RXT_EN
Reset	All zeros									

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	RDR1_EN	RDR0_EN	TDE1_EN	TDE0_EN	ROE1_EN	ROE0_EN	TUE1_EN	TUE0_EN	TFS_EN	RFS_EN	TLS_EN	RLS_EN	RFF1_EN	RFF0_EN	TFE1_EN	TFE0_EN
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Figure 16-23. SSI Interrupt Enable Register (SIER)

Table 16-11 for description of the bit fields in the register.

Table 16-11. SSI Interrupt Enable Register Field Descriptions

Bits	Name	Description
31–25	—	Reserved
24–23	Enable Bits	Enable Bit. Each bit controls whether the corresponding status bit in SISR can issue an interrupt to the Core or not. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.

Table 16-11. SSI Interrupt Enable Register Field Descriptions (continued)

Bits	Name	Description
22	RDMAE	Receive DMA Enable. This bit allows SSI to request for DMA transfers. When enabled, DMA requests are generated when SISR[RFF0] is set and the corresponding SIER[RFF0_EN] is also set. If the FIFO is disabled, a DMA request is generated when the SISR[RDR0] is set. 0 SSI Receiver DMA requests disabled. 1 SSI Receiver DMA requests enabled.
21	RIE	Receive Interrupt Enable. This control bit allows the SSI to issue receiver related interrupts to the Core. Refer to Section 16.4.2, "Receive Interrupt Enable Bit Description," for a detailed description of this bit. 0 SSI Receiver Interrupt requests disabled. 1 SSI Receiver Interrupt requests enabled.
20	TDMAE	Transmit DMA Enable. This bit allows SSI to request for DMA transfers. When enabled, DMA requests are generated when SISR[TFE0] is set and the corresponding SIER[TFE0_EN] is also set. If the FIFO is disabled, a DMA request is generated when SISRTDE0] is set. 0 SSI Transmitter DMA requests disabled. 1 SSI Transmitter DMA requests enabled.
19	TIE	Transmit Interrupt Enable. This control bit allows the SSI to issue transmitter data related interrupts to the Core. Refer to Section 16.4.3, "Transmit Interrupt Enable Bit Description," for a detailed description of this bit. 0 SSI Transmitter Interrupt requests disabled. 1 SSI Transmitter Interrupt requests enabled.
18–0	Enable Bits	Enable Bit. Each bit controls whether the corresponding status bit in SISR can issue an interrupt to the Core or not. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.

16.3.1.10 SSI Transmit Configuration Register (STCR)

The SSI Transmit Configuration Register (STCR) is a read/write control registers used to direct the transmit operation of the SSI. STCR controls the direction of the bit clock and frame sync ports, STCK and STFS. Interrupt enable bit for the transmit sections is provided in this control register. The Power-on reset clears all STCR bits. However, SSI reset does not affect the STCR bits. The STCR bits are described in the following paragraphs. See Table 16-6 for the programming model of the SSI. The SSI Control Register (SCR) must first be set to enable interrupts. Next, the SSI interrupt bit in the Interrupt Enable Register (SIER) must be set to enable the interrupt. Finally, the interrupt can be enabled from within the SSI.

Figure 16-24 shows the SSI transmit configuration register (STCR).

Offset 0x1_601C (STCR) Access: Read/Write

	31–10	9	8	7	6	5	4	3	2	1	0
R W	—	TXBIT0	TFEN1	TFEN0	TFDIR	TXDIR	TSHFD	TSCKP	TFSI	TFSL	TEFS
Reset	0 0000 0000 0000 0000 0000 0000	1	0	0	0	0	0	0	0	0	0

Figure 16-24. SSI Transmit Configuration Register (STCR)

Table 16-12 describes the STCR fields.

Table 16-12. STCR Field Descriptions

Bits	Name	Description
31–10	—	Reserved
9	TXBIT0	Transmit Bit 0. This control bit allows SSI to transmit the data word from bit position 0 or 15/31 in the transmit shift register. The shifting data direction can be msb or lsb first, controlled by the TSHFD bit. 0 Shifting with respect to bit 31 (if word length = 16, 18, 20, 22 or 24) or bit 15 (if word length = 8, 10 or 12) of transmit shift register (msb aligned). 1 Shifting with respect to bit 0 of transmit shift register (lsb aligned).
8	TFEN1	Transmit FIFO Enable 1. This bit enables transmit FIFO 1. When enabled, the FIFO allows 8 samples to be transmitted by the SSI (per channel) (a 9th sample can be shifting out) before TDE1 bit is set. When the FIFO is disabled, an interrupt is generated when a single sample is transferred to the transmit shift register (provided the interrupt is enabled). 0 Transmit FIFO 1 disabled. 1 Transmit FIFO 1 enabled.
7	TFEN0	Transmit FIFO Enable 0. This bit enables transmit FIFO 0. When enabled, the FIFO allows eight samples to be transmitted by the SSI per channel (a 9th sample can be shifting out) before TDE0 bit is set. When the FIFO is disabled, an interrupt is generated when a single sample is transferred to the transmit shift register (provided the interrupt is enabled). 0 Transmit FIFO 0 disabled. 1 Transmit FIFO 0 enabled.
6	TFDIR	Transmit Frame Direction. This bit controls the direction and source of the transmit frame sync signal. Internally generated frame sync signal is sent out through the STFS port and external frame sync is taken from the same port. 0 Frame Sync is external. 1 Frame Sync generated internally.
5	TXDIR	Transmit Clock Direction. This bit controls the direction and source of the clock signal used to clock the TXSR. Internally generated clock is output through the STCK port. External clock is taken from this port. 0 Transmit Clock is external. 1 Transmit Clock generated internally.
4	TSHFD	Transmit Shift Direction. This bit controls whether the msb or lsb will be transmitted first in a sample. **Note:** The CODEC device labels the msb as bit 0, whereas the Core labels the lsb as bit 0. Therefore, when using a standard CODEC, Core msb (CODEC lsb) is shifted in first (TSHFD cleared). 0 Data transmitted msb first. 1 Data transmitted lsb first.
3	TSCKP	Transmit Clock Polarity. This bit controls which bit clock edge is used to clock out data for the transmit section. 0 Data clocked out on rising edge of bit clock. 1 Data clocked out on falling edge of bit clock.
2	TFSI	Transmit Frame Sync Invert. This bit controls the active state of the frame sync I/O signal for the transmit section of SSI. 0 Transmit frame sync is active high. 1 Transmit frame sync is active low.

Table 16-12. STCR Field Descriptions (continued)

Bits	Name	Description
1	TFSL	Transmit Frame Sync Length. This bit controls the length of the frame sync signal to be generated or recognized for the transmit section. The length of a word-long frame sync is same as the length of the data word selected by WL[3:0]. 0 Transmit frame sync is one-word long. 1 Transmit frame sync is one-clock-bit long.
0	TEFS	Transmit Early Frame Sync. This bit controls when the frame sync is initiated for the transmit section. The frame sync signal is deasserted after one bit-for-bit length frame sync and after one word-for-word length frame sync. In case of synchronous operation, the frame sync can also be initiated on receiving the first bit of data. 0 Transmit frame sync initiated as the first bit of data is transmitted. 1 Transmit frame sync is initiated one bit before the data is transmitted.

16.3.1.11 SSI Receive Configuration Register (SRCR)

The SSI receive configuration register (SRCR), shown in Figure 16-25, is a read/write control registers used to direct the receive operation of the SSI. SRCR controls the direction of the bit clock and frame sync ports, SRCK and SRFS. Interrupt enable bit for the transmit sections is provided in this control register. The Power-on reset clears all SRCR bits. However, SSI reset does not affect the SRCR bits.

Offset 0x1_6020 (SRCR) Access: Read/Write

	31–11	10	9	8	7	6	5	4	3	2	1	0
R W	—	RXEXT	RXBIT0	RFEN1	RFEN0	RFDIR	RXDIR	RSHFD	RSCKP	RFSI	RFSL	REFS
Reset	0 0000000000000000000	0	1	0	0	0	0	0	0	0	0	0

Figure 16-25. SSI Receive Configuration Register (SRCR)

Table 16-13 for description of the bit fields in the register.

Table 16-13. SSI Receive Configuration Register Field Descriptions

Bits	Name	Description
31–11	—	Reserved
10	RXEXT	Receive Data Extension. This control bit allows SSI to store the received data word in sign extended form. This bit affects data storage only in case received data is lsb aligned (SRCR[9]=1) 0 Sign extension turned off. 1 Sign extension turned on.
9	RXBIT0	Receive Bit 0. This control bit allows SSI to receive the data word at bit position 0 or 15/31 in the receive shift register. The shifting data direction can be msb or lsb first, controlled by the RSHFD bit. 0 Shifting with respect to bit 31 (if word length = 16, 18, 20, 22 or 24) or bit 15 (if word length = 8, 10 or 12) of receive shift register (msb aligned). 1 Shifting with respect to bit 0 of receive shift register (lsb aligned).
8	RFEN1	Receive FIFO Enable 1. This bit enables receive FIFO 1. When enabled, the FIFO allows eight samples to be received by the SSI per channel (a 9th sample can be shifting in) before RDR1 bit is set. When the FIFO is disabled, an interrupt is generated when a single sample is received by the SSI (provided the interrupt is enabled). 0 Receive FIFO 1 disabled. 1 Receive FIFO 1 enabled.

Table 16-13. SSI Receive Configuration Register Field Descriptions (continued)

Bits	Name	Description
7	RFEN0	Receive FIFO Enable 0. This bit enables receive FIFO 0. When enabled, the FIFO allows 8 samples to be received by the SSI (per channel) (a 9th sample can be shifting in) before RDR0 bit is set. When the FIFO is disabled, an interrupt is generated when a single sample is received by the SSI (provided the interrupt is enabled). 0 Receive FIFO 0 disabled. 1 Receive FIFO 0 enabled.
6	RFDIR	Receive Frame Direction. This bit controls the direction and source of the receive frame sync signal. Internally generated frame sync signal is sent out through the SRFS port and external frame sync is taken from the same port. 0 Frame Sync is external. 1 Frame Sync generated internally.
5	RXDIR	Receive Clock Direction. This bit controls the direction and source of the clock signal used to clock the RXSR. Internally generated clock is output through the SRCK port. External clock is taken from this port. 0 Receive Clock is external. 1 Receive Clock generated internally.
4	RSHFD	Receive Shift Direction. This bit controls whether the msb or lsb will be received first in a sample. **Note:** The CODEC device labels the msb as bit 0, whereas the Core labels the lsb as bit 0. Therefore, when using a standard CODEC, Core msb (CODEC lsb) is shifted in first (RSHFD cleared). 0 Data received msb first. 1 Data received lsb first.
3	RSCKP	Receive Clock Polarity. This bit controls which bit clock edge is used to latch in data for the receive section. 0 Data latched on falling edge of bit clock. 1 Data latched on rising edge of bit clock.
2	RFSI	Receive Frame Sync Invert. This bit controls the active state of the frame sync I/O signal for the receive section of SSI. 0 Receive frame sync is active high. 1 Receive frame sync is active low.
1	RFSL	Receive Frame Sync Length. This bit controls the length of the frame sync signal to be generated or recognized for the receive section. The length of a word-long frame sync is same as the length of the data word selected by WL[3:0]. 0 Receive frame sync is one-word long. 1 Receive frame sync is one-clock-bit long.
0	REFS	Receive Early Frame Sync. This bit controls when the frame sync is initiated for the receive section. The frame sync is disabled after one bit-for-bit length frame sync and after one word-for-word length frame sync. 0 Receive frame sync is initiated one bit before the data is received. 1 Receive frame sync initiated as the first bit of data is received.

16.3.1.12 SSI Transmit and Receive Clock Control Registers (STCCR and SRCCR)

The SSI Transmit and Receive Control (STCCR and SRCCR) registers are used to direct the operation of the SSI. These registers control the SSI clock generator, bit and frame sync rates, word length, and number of words per frame for the serial data. The STCCR register is dedicated to the transmit section, and the SRCCR register is dedicated to the receive section except in Synchronous mode, in which the STCCR register controls both the receive and transmit sections. Power-on reset clears all STCCR and SRCCR bits. SSI reset does not affect the STCCR and SRCCR bits. The control bits are described in the following

paragraphs. Although the bit patterns of the STCCR and SRCCR registers are the same, the contents of these two registers can be programmed differently.

Figure 16-26 shows the SSI transmit and receive clock control registers (STCCR and SRCCR).

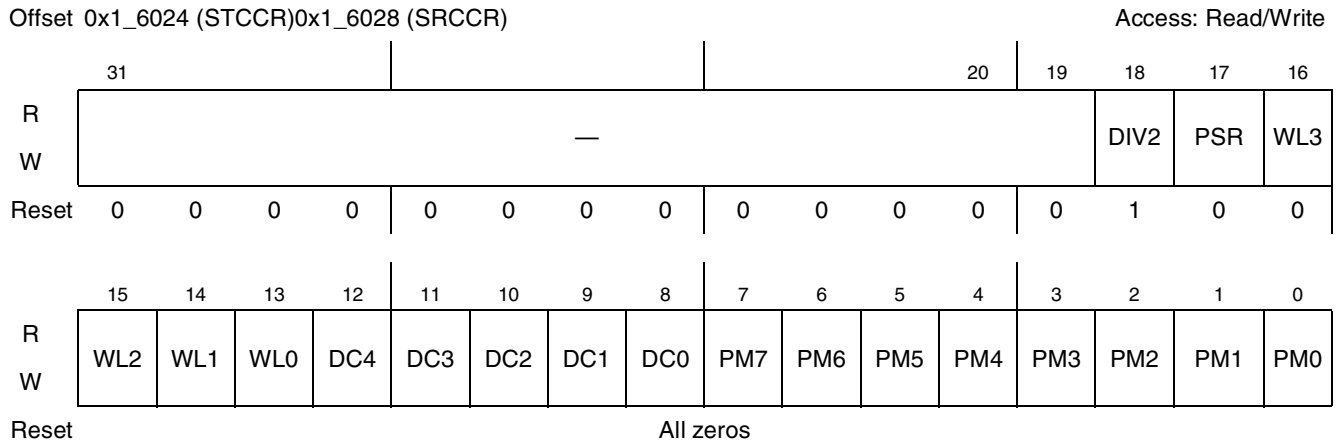


Figure 16-26. SSI Transmit and Receive Clock Control Registers (STCCR and SRCCR)

Table 16-14 for description of the bit fields for both SSI transmit and receive clock control registers.

Table 16-14. SSI Transmit and Receive Clock Control Register Field Descriptions

Bits	Name	Description
31–19	—	Reserved
18	DIV2	Divide By 2. This bit controls a divide-by-two divider in series with the rest of the prescalers. 0 Divider bypassed. 1 Divider used to divide clock by 2.
17	PSR	Prescaler Range. This bit controls a fixed divide-by-eight prescaler in series with the variable prescaler. It extends the range of the prescaler for those cases where a slower bit clock is required. 0 Prescaler bypassed. 1 Prescaler used to divide clock by 8.
16–13	WL*n*	Word length control. These bits are used to control the length of the data words being transferred by the SSI. These bits control the word length divider in the clock generator. They also control the frame sync pulse length when the FSL bit is cleared. In I2S Master mode, the SSI works with a fixed word length of 32, and the WL bits are used to control the amount of valid data in those 32 bits. See Table 16-15 for details of data word lengths supported by SSI. In AC97 mode of operation, if word length is set to any value other than 16 bits, it will result in a word length of 20 bits.

Table 16-14. SSI Transmit and Receive Clock Control Register Field Descriptions (continued)

Bits	Name	Description
12–8	DC*n*	Frame Rate Divider Control. These bits are used to control the divide ratio for the programmable frame rate dividers. The divide ratio works on the word clock. In Normal mode, this ratio determines the word transfer rate. In Network mode, this ratio sets the number of words per frame. The divide ratio ranges from 1 to 32 in Normal mode and from 2 to 32 in Network mode. In Normal mode, a divide ratio of 1 (DC=00000) provides continuous periodic data word transfer. A bit-length frame sync must be used in this case. These bits can be programmed with values ranging from “00000” to “11111” to control the number of words in a frame.
7–0	PM*n*	Prescaler modulus select. These bits control the prescale divider in the clock generator. This prescaler is used only in Internal Clock mode to divide the internal clock (*ccm_ssi_clk*). The bit clock output is available at the clock port. A divide ratio from 1 to 256 (PM[7:0] = 0x00 to 0xFF) can be selected. See Section 16.4.1.2, “DIV2, PSR and PM Bit Description,” for details regarding settings.

Table 16-15. SSI Data Length

WL3	WL2	WL1	WL0	Number of Bits/Word	Supported in Implementation
0	0	0	0	2	No
0	0	0	1	4	No
0	0	1	0	6	No
0	0	1	1	8	Yes
0	1	0	0	10	Yes
0	1	0	1	12	Yes
0	1	1	0	14	No
0	1	1	1	16	Yes
1	0	0	0	18	Yes
1	0	0	1	20	Yes
1	0	1	0	22	Yes
1	0	1	1	24	Yes
1	1	0	0	26	No
1	1	0	1	28	No
1	1	1	0	30	No
1	1	1	1	32	No

16.3.1.13 SSI FIFO Control/Status Register (SFCSR)

Figure 16-27 shows the SSI FIFO control/status register (SFCSR).

Offset 0x1_602C (SFCSR) Access: Read/Write

	31–28	27–24	23–20	19–16	15–12	11–8	7–4	3–0
R W	RFCNT1	TFCNT1	RFWM1	TFWM1	RFCNT0	TFCNT0	RFWM0	TFWM0
Reset	1 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 1	0 0 0 0	0 0 0 0

Figure 16-27. SSI FIFO Control/Status Register (SFCSR)

Table 16-16 describes the bit fields in SFCSR.

Table 16-16. SSI FIFO Control/Status Register Field Descriptions

Bits	Name	Description
31–28	RFCNT1	Receive FIFO counter 1. This field indicates the number of data words in receive FIFO 1; values greater than 1000 are invalid.
27–24	TFCNT1	Transmit FIFO counter 1. This field indicates the number of data words in transmit FIFO 1; values greater than 1000 are invalid.
22–20	RFWM1	Receive FIFO full watermark 1. This field controls the threshold at which the RFF1 flag will be set. The RFF1 flag is set whenever the data level in Rx FIFO 1 reaches the selected threshold. 0000 Reserved 0001 RFF set when at least one data word has been written to the receive FIFO Set when RxFIFO = 1,2,3,4,5,6,7,8 data words 0010 RFF set when more than or equal to 2 data word have been written to the receive FIFO. Set when RxFIFO = 2,3,4,5,6,7,8 data words 0011 RFF set when more than or equal to 3 data word have been written to the receive FIFO. Set when RxFIFO = 3,4,5,6,7,8 data words 0100 RFF set when more than or equal to 4 data word have been written to the receive FIFO. Set when RxFIFO = 4,5,6,7,8 data words 0101 RFF set when more than or equal to 5 data word have been written to the receive FIFO. Set when RxFIFO = 5,6,7,8 data words 0110 RFF set when more than or equal to 6 data word have been written to the receive FIFO. Set when RxFIFO = 6,7,8 data words 0111 RFF set when more than or equal to 7 data word have been written to the receive FIFO. Set when RxFIFO = 7,8 data words 1000 RFF set when 8 data word have been written to the receive FIFO (default). Set when RxFIFO = 8 data words

Table 16-16. SSI FIFO Control/Status Register Field Descriptions (continued)

Bits	Name	Description
19–16	TFWM1	Transmit FIFO empty watermark 1. This field controls the threshold at which the TFE1 flag will be set. The TFE1 flag is set whenever the data level in Tx FIFO 1 falls below the selected threshold. 0000 Reserved 0001 TFE set when there are more than or equal to 1 empty slots in transmit FIFO. (default) Transmit FIFO empty is set when TxFIFO <= 7 data. 0010 TFE set when there are more than or equal to 2 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 6 data. 0011 TFE set when there are more than or equal to 3 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 5 data. 0100 TFE set when there are more than or equal to 4 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 4 data. 0101 TFE set when there are more than or equal to 5 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 3 data. 0110 TFE set when there are more than or equal to 6 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 2 data. 0111 TFE set when there are more than or equal to 7 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 1 data. 1000 TFE set when there are 8 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO=0 data.
15–12	RFCNT0	Receive FIFO counter 0. This field indicates the number of data words in receive FIFO 0; values greater than 1000 are invalid.
11–8	TFCNT0	Transmit FIFO counter 0. This field indicates the number of data words in transmit FIFO 0; values greater than 1000 are invalid.

Table 16-16. SSI FIFO Control/Status Register Field Descriptions (continued)

Bits	Name	Description
7–4	RFWM0	Receive FIFO full watermark 0. This field controls the threshold at which the RFF0 flag will be set. The RFF0 flag is set whenever the data level in Rx FIFO 0 reaches the selected threshold. 0000 Reserved 0001 RFF set when at least one data word has been written to the receive FIFO Set when RxFIFO = 1,2,3,4,5,6,7,8 data words 0010 RFF set when more than or equal to 2 data word have been written to the receive FIFO. Set when RxFIFO = 2,3,4,5,6,7,8 data words 0011 RFF set when more than or equal to 3 data word have been written to the receive FIFO. Set when RxFIFO = 3,4,5,6,7,8 data words 0100 RFF set when more than or equal to 4 data word have been written to the receive FIFO. Set when RxFIFO = 4,5,6,7,8 data words 0101 RFF set when more than or equal to 5 data word have been written to the receive FIFO. Set when RxFIFO = 5,6,7,8 data words 0110 RFF set when more than or equal to 6 data word have been written to the receive FIFO. Set when RxFIFO = 6,7,8 data words 0111 RFF set when more than or equal to 7 data word have been written to the receive FIFO. Set when RxFIFO = 7,8 data words 1000 RFF set when 8 data word have been written to the receive FIFO (default). Set when RxFIFO = 8 data words **Note:** If DMA requests are used, the value of this field must be > 8 – (DMA transfer size)/4, and it must also be ≥ (DMA transfer size)/4. For example, with DMA transfer size of 16 bytes (4 slots), SFCSR[RFWM0] ≥ 5, and with DMA transfer size of 8 bytes (2 characters), SFCSR[RFWM0] ≥ 7.
3–0	TFWM0	Transmit FIFO empty watermark 0. This field controls the threshold at which the TFE0 flag will be set. The TFE0 flag is set whenever the data level in Tx FIFO 0 falls below the selected threshold. 0000 Reserved 0001 TFE set when there are more than or equal to 1 empty slots in transmit FIFO. (default) Transmit FIFO empty is set when TxFIFO <= 7 data. 0010 TFE set when there are more than or equal to 2 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 6 data. 0011 TFE set when there are more than or equal to 3 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 5 data. 0100 TFE set when there are more than or equal to 4 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 4 data. 0101 TFE set when there are more than or equal to 5 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 3 data. 0110 TFE set when there are more than or equal to 6 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 2 data. 0111 TFE set when there are more than or equal to 7 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 1 data. 1000 TFE set when there are 8 empty slots in transmit FIFO. Transmit FIFO empty is set when TxFIFO=0 data. **Note:** If DMA requests are used, the value of this field must be greater than the (DMA transfer size)/4. For example, with DMA transfer size of 16 bytes (4 slots), SFCSR[TFWM0] > 4.

Table 16-17 indicates the status of the transmit FIFO empty flag, with different settings of the transmit FIFO watermark bits and varying amounts of data in the Tx FIFO.

Table 16-17. Status of Transmit FIFO Empty Flag

Transmit FIFO Watermark (TFWM)	Number of data in Tx-Fifo							
	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	0
2	1	1	1	1	1	1	0	0
3	1	1	1	1	1	0	0	0
4	1	1	1	1	0	0	0	0
5	1	1	1	0	0	0	0	0
6	1	1	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

16.3.1.14 SSI AC97 Control Register (SACNT)

Figure 16-28 shows the SSI AC97 control register (SACNT).

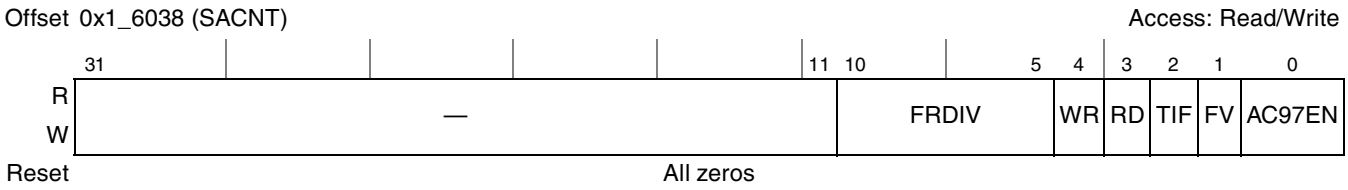


Figure 16-28. SSI AC97 Control Register (SACNT)

Table 16-18 describes the SACNT fields.

Table 16-18. SACNT Field Descriptions

Bits	Name	Description
31–11	—	Reserved
10–5	FRDIV	Frame Rate Divider. These bits control the frequency of AC97 data transmission/reception. They are programmed with the number of frames for which the SSI should be idle, after operating in one frame. Through these bits, AC97 frequency of operation, from 48 KHz (000000) to 1 KHz (101111) can be achieved. Sample Value: 001010 (10 Decimal) = SSI will operate once every 11 frames.
4	WR	Write Command. This bit specifies whether the next frame will carry an AC97 Write Command or not. The programmer should take care that only one of the bits (WR or RD) is set at a time. When this bit is set, the corresponding tag bits (corresponding to Command Address and Command Data slots of the next Tx frame) are automatically set. This bit is automatically cleared by the SSI after completing transmission of a frame. 0 Next frame will not have a Write Command. 1 Next frame will have a Write Command.

Table 16-18. SACNT Field Descriptions (continued)

Bits	Name	Description
3	RD	Read Command. This bit specifies whether the next frame will carry an AC97 Read Command or not. The programmer should take care that only one of the bits (WR or RD) is set at a time. When this bit is set, the corresponding tag bit (corresponding to Command Address slot of the next Tx frame) is automatically set. This bit is automatically cleared by the SSI after completing transmission of a frame. 0 Next frame will not have a Read Command. 1 Next frame will have a Read Command.
2	TIF	Tag in FIFO. This bit controls the destination of the information received in AC97 tag slot (Slot #0). 0 Tag info stored in SATAG register. 1 Tag info stored in Rx FIFO 0.
1	FV	Fixed/Variable Operation. This bit selects whether the SSI is in AC97 Fixed mode or AC97 Variable mode. 0 AC97 Fixed Mode. 1 AC97 Variable Mode.
0	AC97EN	AC97 Mode Enable. This bit is used to enable SSI AC97 operation. Refer to Section 16.1.2.4 for details of AC97 operation. 0 AC97 mode disabled. 1 SSI in AC97 mode.

16.3.1.15 SSI AC97 Command Address Register (SACADD)

Figure 16-29 shows the SSI AC97 command address register (SACADD).

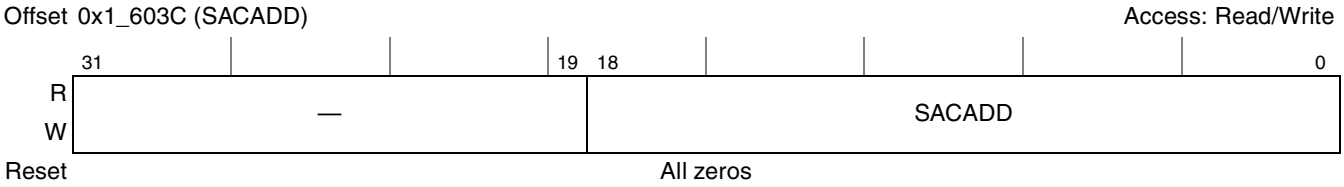


Figure 16-29. SSI AC97 Command Address Register (SACADD)

Table 16-19 describes the SACADD fields.

Table 16-19. SACADD Field Descriptions

Bits	Name	Description
31–19	—	Reserved
18–0	SACADD	AC97 Command Address. These bits store the Command Address Slot information (bit 19 of the slot is sent in accordance with the Read and Write Command bits in SACNT register). These bits can be updated by a direct write from the Core. They are also updated with the information received in the incoming Command Address Slot. If the contents of these bits change due to an update, the CMDAU bit in SISR is set.

16.3.1.16 SSI AC97 Command Data Register (SACDAT)

Figure 16-30 shows the SSI AC97 command data register (SACDAT).

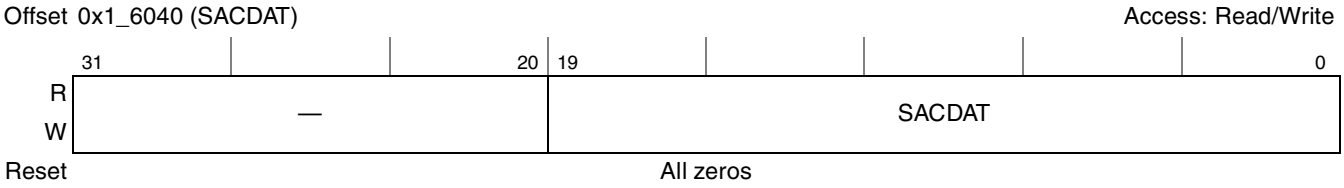


Figure 16-30. SSI AC97 Command Data Register (SACDAT)

Table 16-20 describes the SACDAT fields.

Table 16-20. SACDAT Field Descriptions

Bits	Name	Description
31–20	—	Reserved
19–0	SACDAT	AC97 Command Data. The outgoing Command Data Slot carries the information contained in these bits. These bits can be updated by a direct write from the Core. They are also updated with the information received in the incoming Command Data Slot. If the contents of these bits change due to an update, the CMDDU bit in SISR is set. These bits are transmitted only during AC97 Write Command. During AC97 Read Command, 0x00000 is transmitted in time slot #2.

16.3.1.17 SSI AC97 Tag Register (SATAG)

Figure 16-31 shows the SSI AC97 tag register (SATAG).

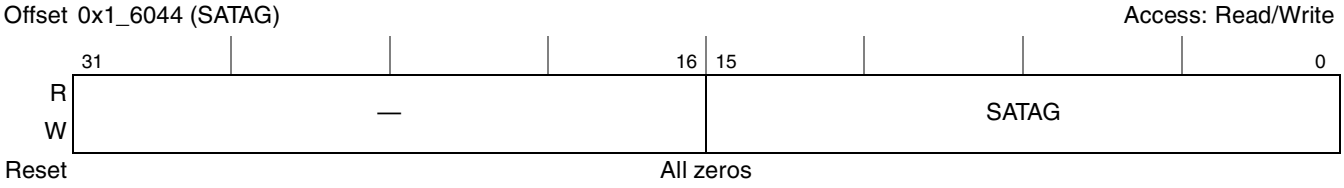


Figure 16-31. SSI AC97 Tag Register (SATAG)

Table 16-21 for description of the bit fields for the register.

Table 16-21. SSI AC97 Tag Register Field Descriptions

Bits	Name	Description
31–16	—	Reserved
15–0	SATAG	AC97 Tag Value. Writing to this register (by the Core) sets the value of the Tx-Tag in AC97 fixed mode of operation. On a read, the Core gets the Rx-Tag Value received (in the last frame) from the Codec. If TIF bit in SACNT register is set, the TAG value is also stored in Rx-FIFO in addition to SATAG register. When the received Tag value changes, the RXT bit in SISR register is set. Bits SATAG[1:0] convey the Codec -ID. In current implementation only Primary Codecs are supported. Thus writing value 2'b00 to this field is mandatory.

16.3.1.18 SSI Transmit Time Slot Mask Register (STMSK)

Figure 16-32 shows the SSI transmit time slot mask register (STMSK).

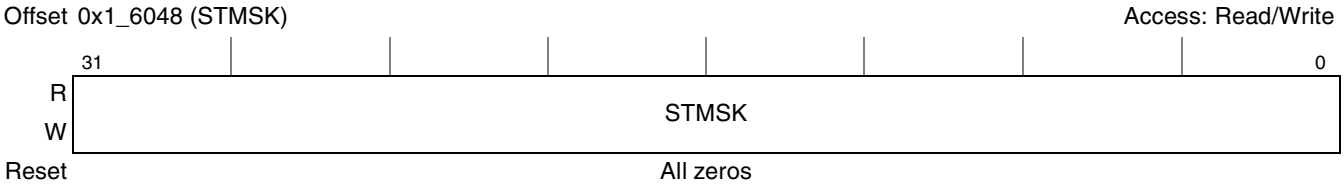


Figure 16-32. SSI Transmit Time Slot Mask Register (STMSK)

Table 16-22 describes the STMSK fields.

Table 16-22. STMSK Field Descriptions

Bits	Name	Description
31–0	STMSK	Transmit Mask. These bits indicate which slot has been masked in the current frame. The Core can write to this register to control the time slots in which the SSI transmits data. Each bit has info corresponding to the respective time slot in the frame. If a change is made to the register contents, the transmission pattern is updated from the next time slot. Transmit mask bits should not be used in I2S Slave mode of operation. 0 Valid Time Slot. 1 Time Slot masked (no data transmitted in this time slot).

16.3.1.19 SSI Receive Time Slot Mask Register (SRMSK)

Figure 16-33 shows the SSI receive time slot mask register (SRMSK).

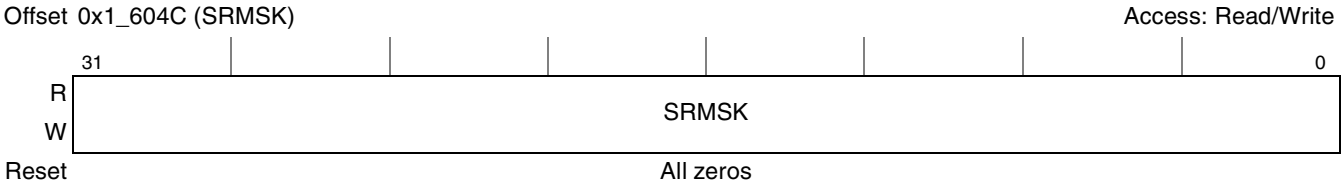


Figure 16-33. SSI Receive Time Slot Mask Register (SRMSK)

Table 16-23 describes the SRMSK fields.

Table 16-23. SRMSK Field Descriptions

Bits	Name	Description
31–0	SRMSK	Receive Mask. These bits indicate which slot has been masked in the current frame. The Core can write to this register to control the time slots in which the SSI receives data. Each bit has info corresponding to the respective time slot in the frame. If a change is made to the register contents, the reception pattern is updated from the next time slot. Receive mask bits should not be used in I2S Slave mode of operation. 0 Valid Time Slot. 1 Time Slot masked (no data received in this time slot).

16.3.1.20 SSI AC97 Channel Status Register (SACCST)

Figure 16-34 shows the SSI AC97 channel status register (SACCST).

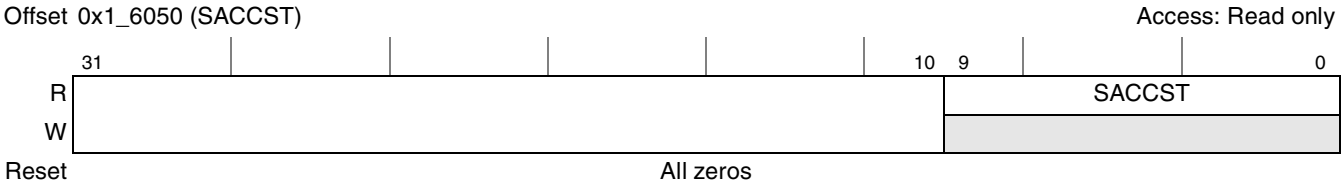


Figure 16-34. SSI AC97 Channel Status Register (SACCST)

Table 16-24 describes the SACCST fields.

Table 16-24. SACCST Field Descriptions

Bits	Name	Description
31–10	—	Reserved
9–0	SACCST	AC97 Channel Status. These bits indicate which data slot has been enabled in AC97 variable mode operation. This register is updated in case the core enables/disables a channel through a write to SACCEN/sACCDIS register or the external codec enables a channel by sending a ‘1’ in the corresponding SLOTREQ bit. Bit [0] corresponds to the first data slot in an AC97 frame (Slot #3) and Bit [9] corresponds to the tenth data slot (slot #12). The contents of this register only have relevance while the SSI is operating in AC97 variable mode. Writes to this register result in an error response on the IP interface. 0 Data channel disabled. 1 Data channel enabled.

16.3.1.21 SSI AC97 Channel Enable Register (SACCEN)

Figure 16-35 shows the SSI AC97 channel enable register (SACCEN).

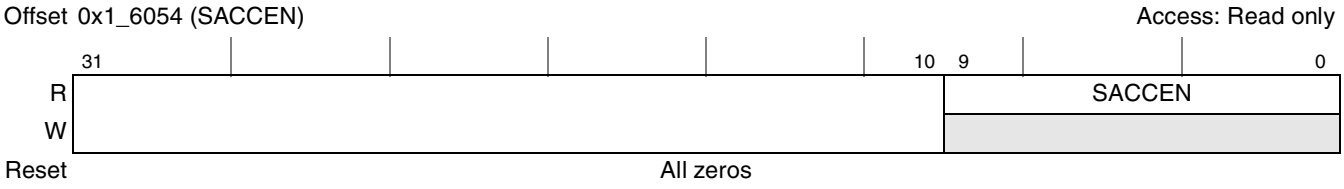


Figure 16-35. SSI AC97 Channel Enable Register (SACCEN

Table 16-26 describes the SACCEN fields.

Table 16-25. SACCEN Field Descriptions

Bits	Name	Description
31–10	—	Reserved
9–0	SACCEN	AC97 Channel Enable. The Core writes a ‘1’ to these bits to enable an AC97 data channel. Writing a ‘0’ has no effect. Bit [0] corresponds to the first data slot in an AC97 frame (Slot #3) and Bit [9] corresponds to the tenth data slot (slot #12). Writes to these bits only have effect in the AC97 Variable mode of operation. These bits are always read as ‘0’ by the Core. 0 Write Has no effect. 1 Write Enables the corresponding data channel.

16.3.1.22 SSI AC97 Channel Disable Register (SACCDIS)

Figure 16-36 shows the SSI AC97 channel disable register (SACCDIS).

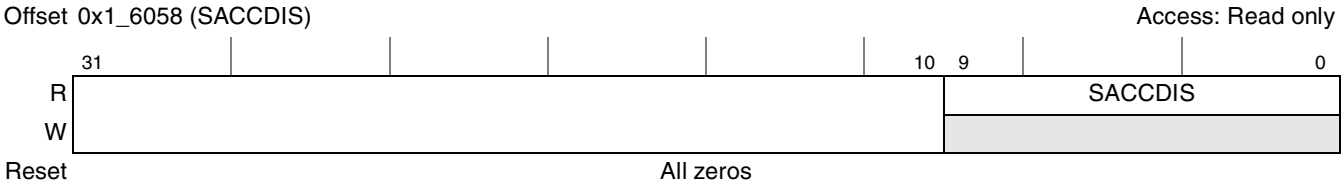


Figure 16-36. SSI AC97 Channel Disable Register (SACCDIS)

Table 16-26 describes the SACCDIS fields.

Table 16-26. SACCDIS Field Descriptions

Bits	Name	Description
31–10	—	Reserved
9–0	SACCDIS	AC97 Channel Disable. The Core writes a '1' to these bits to disable an AC97 data channel. Writing a '0' has no effect. Bit [0] corresponds to the first data slot in an AC97 frame (Slot #3) and Bit [9] corresponds to the tenth data slot (slot #12). Writes to these bits only have effect in the AC97 Variable mode of operation. These bits are always read as '0' by the Core. 0 Write Has no effect. 1 Write Disables the corresponding data channel.

16.4 SSI Functional Description

16.4.1 SSI Clocking

The SSI uses the following clocks:

- Bit clock — Used to serially clock the data bits in and out of the SSI port. This clock is either generated internally (from *ccm_ssi_clk*) or taken from external clock source (through the Tx/Rx clock ports).
- Word clock — Used to count the number of data bits per word (8, 10, 12, 16, 18, 20, 22 or 24 bits). This clock is generated internally from the bit clock.
- Frame clock (Frame Sync) — Used to count the number of words in a frame. This signal can be generated internally from the bit clock, or taken from external source (from the Tx/Rx frame sync ports).
- SSI clock —This is *ccm_ssi_clk*. The SSI clock is controlled by the global utilities CLKDVDR register (refer to Section 23.4.1.19, "Clock Divide Register (CLKDVDR)," for more information). CLKDVDR[ssicken] enables the SSI clock and CLKDVDR[ssiclk] sets the platform clock divisor used to generate the SSI clock (*ccm_ssi_clk*). In master mode, SSI clock is an integer multiple of frame clock. It is used in cases when SSI has to provide the clock.

Care should be taken to ensure that the bit clock frequency (either internally generated by dividing the *ccm_ssi_clk* or sourced from external device through Tx/Rx clock ports) is never greater than 1/5 of the platform clock frequency.

In Normal mode (SCR[6:5]=00), the bit clock, used to serially clock the data, is visible on the Serial Transmit Clock (STCK) and Serial Receive Clock (SRCK) ports. The word clock is an internal clock used to determine when transmission of an 8, 10, 12, 16, 18, 20, 22 or 24 bit word has completed. The word clock in turn then clocks the frame clock, which counts the number of words in the frame. The frame clock can be viewed on the STFS and SRFS frame sync ports, because a frame sync is generated after the correct number of words in the frame have passed. The word length (WL), Prescaler Range (PSR), Prescaler Modulus (PM) and Frame rate (DC) selects the ratio of the SSI clock to sampling clock STFS. The relationship between the clocks and the dividers is shown in Figure 16-37 (“SSI Clocking”). The bit clock can be received from an SSI clock port or can be generated from the *ccm_ssi_clk* through a divider, as shown in Figure 16-38 (“SSI Transmit Clock Generator Block Diagram”).

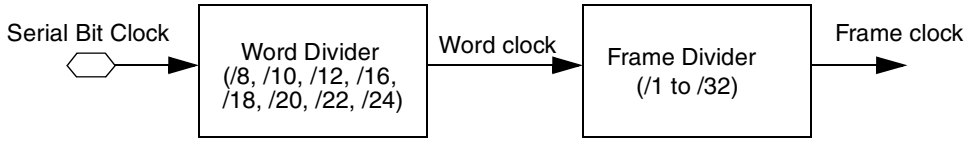


Figure 16-37. SSI Clocking

16.4.1.1 SSI Clock and Frame Sync Generation

Data clock and frame sync signals can be generated internally, or can be obtained from external sources. If internally generated, the SSI clock generator is used to derive bit clock and frame sync signals from the *ccm_ssi_clk* clock. The SSI clock generator consists of a selectable, fixed prescaler and a programmable prescaler for bit rate clock generation. A programmable frame rate divider and a word length divider are used for frame rate sync signal generation.

Figure 16-38 shows a block diagram of the clock generator for the transmit section. The serial bit clock can be internal or external, depending on the Transmit Direction (TXDIR) bit in the SSI Transmit Configuration Register (STCR). The receive section contains an equivalent clock generator circuit.

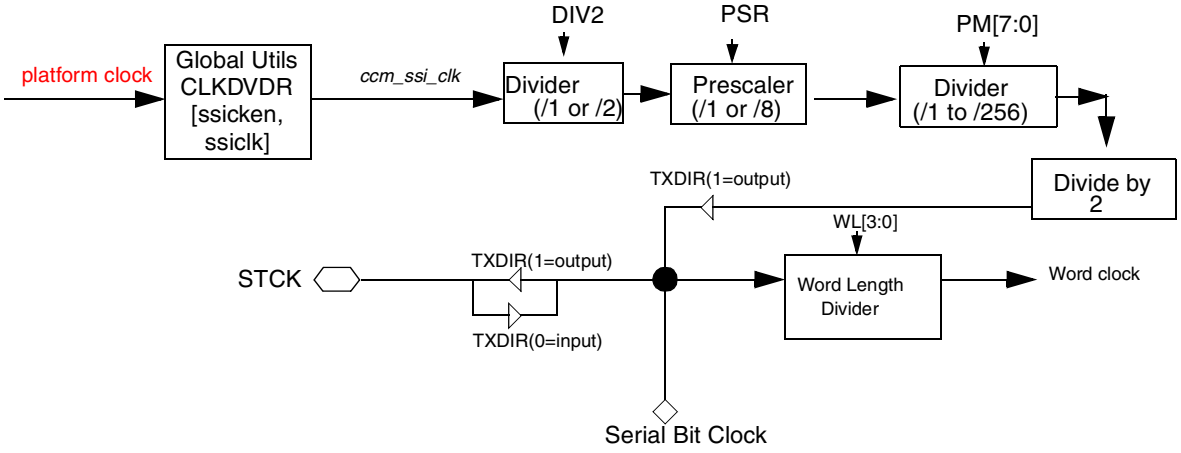


Figure 16-38. SSI Transmit Clock Generator Block Diagram

See Figure 16-39 shows the Frame Sync Generator block for the transmit section. When internally generated, both receive and transmit frame sync are generated from the word clock and are defined by the Frame Rate Divider (DC[4:0]) bits and the Word Length (WL[3:0]) bits of the SSI Transmit Clock Control Register (STCCR). The receive section contains an equivalent circuit for the Frame Sync Generator.

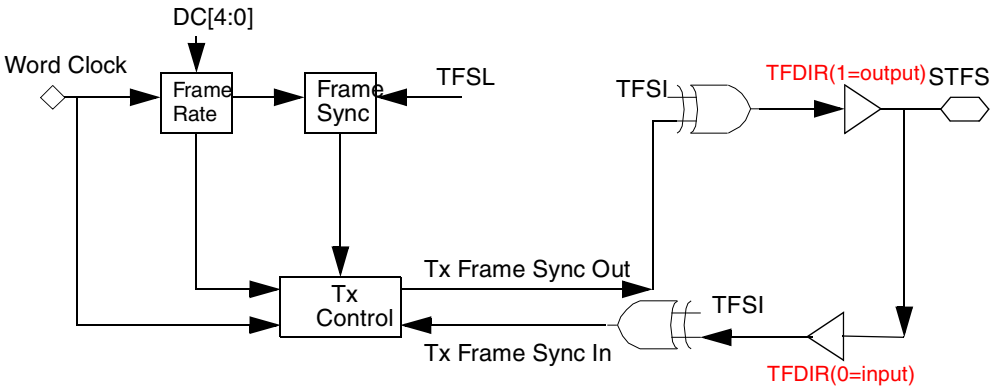


Figure 16-39. SSI Transmit Frame Sync Generator Block Diagram

16.4.1.2 DIV2, PSR and PM Bit Description

The bit clock frequency can be calculated from the SSI Serial System Clock (*ccm_ssi_clk*) using the equation in Figure 16-40.

NOTE

The bit-clock frequency must be less than 1/5 the platform clock frequency. The oversampling clock frequency can go up to the platform clock frequency. Bits DIV2, PSR and PM should not be all set to zero at the same time.

$$f_{INT\_BIT\_CLK} = f_{SYS\_CLK} / [(DIV2 + 1) \times (7 \times PSR + 1) \times (PM + 1) \times 2]$$

where PM=PM[7:0]

$$f_{FRAME\_SYN\_CLK} = (f_{INT\_BIT\_CLK}) / [(DC + 1) \times WL]$$

where DC = DC[4:0] and WL = 8, 10, 12, 16, 18, 20, 22, 24

Figure 16-40. SSI Bit Clock Equation

For example, if the SSI working clock (*ccm_ssi_clk*) is 12.288, in 8-bit word Normal mode with DC[4:0] set to 1 (00001), PM[7:0] set to 47 (0010 1111), the PSR bit cleared, DIV2 bit set to 1, a bit clock rate of 12.288 MHz / [1 x 4 x 48] = 64 kHz is generated. Since the 8-bit word rate is equal to one (i.e. normal mode), the sampling rate (FS rate) would then be 64 kHz / [1 * 8] = 8 kHz.

In next example, *ccm_ssi_clk* is 11.2896 MHz. A 16-bit word Network mode with DC[4:0] set to 1 (00001), PM[7:0] set to 3 (0000 0011), the PSR bit is set to 0, DIV2 bit set to 0, and a 11.2896 MHz

SYS_CLK clock, a bit clock rate of 11.2896 MHz / [1 x 2 x 4] = 1.4112 MHz is generated. Since the 16-bit word rate is equal to two, the sampling rate (FS rate) would be 1.4112 MHz / [2 * 16] = 44.1 kHz.

Table 16-27 shows programming examples for the clock dividers in the CRM and the SSI to support various bit clock (STCK) frequencies.

Table 16-27. SSI Bit Clock and Frame Rate as a Function of PSR, PM, and DIV2

Bits/ Word	Words/ Frame	Ideal Frame Rate (kHz)	Platform Freq (MHz)	CLKDVDR [SSICLK]	*ccm_ssi_clk* Freq (MHz)	DIV 2	PS R	PM	WL	DC	Actual Bit_Clk Freq (kHz) STCK	Target Bit_Clk Freq (kHz) STCK	Error (Hz)
16	1	8	589.824	48	12.288	0	0	47	7	0	128	128	0
16	2	8	589.824	48	12.288	0	0	23	7	1	256	256	0
16	4	8	589.824	48	12.288	0	0	11	7	3	512	512	0
16	1	12	589.824	48	12.288	0	0	31	7	0	192	192	0
16	2	12	589.824	48	12.288	0	0	15	7	1	384	384	0
16	4	12	589.824	48	12.288	0	0	7	7	3	768	768	0
16	1	16	589.824	48	12.288	0	0	23	7	0	256	256	0
16	2	16	589.824	48	12.288	0	0	11	7	1	512	512	0
16	4	16	589.824	48	12.288	0	0	5	7	3	1024	1024	0
16	1	24	589.824	48	12.288	0	0	15	7	0	384	384	0
16	2	24	589.824	48	12.288	0	0	7	7	1	768	768	0
16	4	24	589.824	48	12.288	0	0	3	7	3	1536	1536	0
16	1	32	589.824	48	12.288	0	0	11	7	0	512	512	0
16	2	32	589.824	48	12.288	0	0	5	7	1	1024	1024	0
16	4	32	589.824	48	12.288	0	0	2	7	3	2048	2048	0
16	1	48	589.824	48	12.288	0	0	15	7	0	768	768	0
16	2	48	589.824	48	12.288	0	0	3	7	1	1536	1536	0
16	4	48	589.824	48	12.288	0	0	1	7	3	3072	3072	0
16	1	11.025	541.9008	48	11.2896	0	0	31	7	0	176.4	176.4	0
16	2	11.025	541.9008	48	11.2896	0	0	15	7	1	352.8	352.8	0
16	4	11.025	541.9008	48	11.2896	0	0	7	7	3	705.6	705.6	0
16	1	22.05	541.9008	48	11.2896	0	0	15	7	0	352.8	352.8	0
16	2	22.05	541.9008	48	11.2896	0	0	7	7	1	705.6	705.6	0
16	4	22.05	541.9008	48	11.2896	0	0	3	7	3	1411.2	1411.2	0
16	1	44.1	541.9008	48	11.2896	0	0	7	7	0	705.6	705.6	0

Table 16-27. SSI Bit Clock and Frame Rate as a Function of PSR, PM, and DIV2 (continued)

Bits/ Word	Words/ Frame	Ideal Frame Rate (kHz)	Platform Freq (MHz)	CLKDVDR [SSICLK]	*ccm_ssi_clk* Freq (MHz)	DIV 2	PS R	PM	WL	DC	Actual Bit_Clk Freq (kHz) STCK	Target Bit_Clk Freq (kHz) STCK	Error (Hz)
16	2	44.1	541.9008	48	11.2896	0	0	3	7	1	1411.2	1411.2	0
16	4	44.1	541.9008	48	11.2896	0	0	1	7	3	2822.4	2822.4	0

Table 16-28 below shows programming of the CRM and SSI dividers in order to generate the appropriate SSI clock (*ccm_ssi_clk*) and BIT_CLK frequencies for various sampling rates. In these examples, the master mode is selected either by setting I2S master bit (SCR[6:5]=01) or individually programming the SSI in network, synchronous, transmit internal mode (the table specifically illustrates the I2S mode frequencies/sample rates).

Note that the I2S master mode requires that a word length of 32 bits be used (regardless of the actual data type). Consequently, the fixed I2S frame rate of 64 bits per frame (word length (WL) can be any value) and DC of 1 are assumed.

Table 16-28. SSI Sys Clock, Bit Clock, Frame Clock in Master Mode

Bits / Word	Words / Frame	Ideal Sampling Rate (kHz)	Over Sampling Rate	Platform Freq (MHz)	CLKDVDR [SSICLK]	Target ccm_ssi _clk Freq (MHz)	Actual ccm_ssi _clk Freq (MHz)	Bits / Word	Words / Frame	Actual Sampling Rate (kHz) STFS	Error (Hz)
32	2	22.05	512	541.9004	48	11.2896	11.2896	32	2	44.117	17.65
32	2	24	512	589.824	48	12.288	12.288	32	2	22.058	8.82
32	2	32	384	589.824	48	12.288	12.288	32	2	11.029	4.41
32	2	32	512	589.824	36	16.384	16.384	32	2	48.000	0

16.4.2 Receive Interrupt Enable Bit Description

When the RIE and RE bit are set, the processor is interrupted when either of the SSI Receive FIFO Full (RFF0/1) bits in SISR is set (if the corresponding Receive FIFO is enabled). If the Receive FIFO is not enabled, the interrupt is generated when the corresponding SSI Receive Data Ready (RDR0/1) bit in the SISR is set. When the receive FIFO is enabled, a maximum of eight values are available to be read (8 values per channel in two-channel mode). If not enabled, then one value can be read from the SRX register (one each in case of two-channel mode). If the RIE bit is cleared, these interrupts are disabled. However, the RFF0/1 and RDR0/1 bits still indicate the receive data register full condition. Reading the SRX registers clears the RDR bits, thus clearing the pending interrupt. Two receive data interrupts (two per channel in case of two-channel mode) are available: receive data with exception status and receive data without exception. Table 16-29 and Table 16-30 show the conditions under which these interrupts are generated.

Table 16-29. SSI Receive Data 1 Interrupts

Interrupt	RIE	ROE0	RFF0/RDR0
Receive Data 1(with Exception Status)	1	1	1
Receive Data 1(without exception)	1	0	1

Table 16-30. SSI Receive Data 0 Interrupts

Interrupt	RIE	ROE1	RFF1/RDR1
Receive Data 0 (with Exception Status)	1	1	1
Receive Data 0 (without exception)	1	0	1

16.4.3 Transmit Interrupt Enable Bit Description

The SSI Transmit Interrupt Enable (TIE) control bit determines whether the processor is interrupted when the SSI transmitter needs to be serviced. When the TIE and TE bits are set, the program controller is interrupted when either of the SSI Transmit FIFO Empty (TFE0/1) flags in SISR are set (if corresponding Transmit FIFO is enabled). If the corresponding Transmit FIFO is not enabled, an interrupt is generated when the corresponding SSI Transmit Data Register Empty (TDE0/1) flag in the SISR is set and Transmit Enable (TE) bit is set.

When Transmit FIFO 0 is enabled, a maximum of eight values can be written to the SSI (8 per channel in case of two-channel mode, using Tx FIFO 1). If not enabled, then one value can be written to the STX0 register (one per channel in case of two-channel mode using STX1). When the TIE bit is cleared, all transmit interrupts are disabled. However, the TDE0/1 bits always indicate the corresponding STX register empty condition, even when the transmitter is disabled by the Transmit Enable (TE) bit (in the SCR). Writing data to the STX clears the corresponding TDE bit, thus clearing the interrupt. Two transmit data interrupts are available (four in case of two-channel mode, two per channel): transmit data with exception status and transmit data without exceptions. Table 16-31 and Table 16-32 show the conditions under which these interrupts are generated.

Table 16-31. SSI Transmit Data 1 Interrupts

Interrupt	TIE	TUE1	TFE1/TDE1
Transmit Data 1 (with Exception Status)	1	1	1
Transmit Data 1 (without exception)	1	0	1

Table 16-32. SSI Transmit Data 0 Interrupts

Interrupt	TIE	TUE0	TFE0/TDE0
Transmit Data 0 (with Exception Status)	1	1	1
Transmit Data 0 (without exception)	1	0	1

16.4.4 Internal Frame and Clock Shutdown

During transmit/receive operation, disabling TE/RE will ensure that data transmission/reception stops after current frame ends following which TFRC/RFRC Status bits will get set to indicate the Frame Completion State. If TFR_CLK_DIS/RFR_CLK_DIS bit is set in the current or any of the previous frames, SSI will stop driving the STFS/SRFS and STCK/SRCK signals after the current frame ends.

If TFR_CLK_DIS/RFR_CLK_DIS bit is not set, SSI will continue generating STFS/SRFS and STCK/SRCK signals (in case direction is from SSI), which then can be disabled by writing ‘1’ to TFS_CLK_DIS/RFR_CLK_DIS bit. SSI will then stop driving these signals after end of frame is reached following which TFRC/RFRC status bits will get set to indicate the Frame Completion State.

Figure 16-41 is an illustration of transmission case where TXDIR and TFDIR are both set to ‘1’. In this case TE is disabled with TFS_CLK_DIS bit set in current or any of the previous frames.

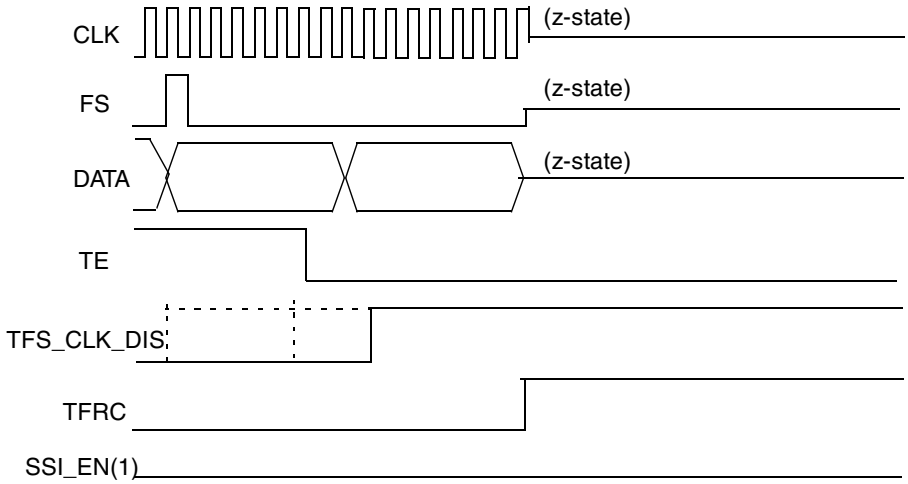


Figure 16-41. TFS_CLK_DIS Assertion in Current or Previous Frame as TE Disable

Figure 16-42 is an illustration of transmission case where TXDIR and TFDIR are both set to ‘1’. In this case TFS_CLK_DIS bit is set after few frames of disabling TE. TFRC (Transmit Frame Complete) is set

at frame boundary after TE is cleared. Once software services this interrupt and sets TFR_CLK_DIS bit later, TFRC bit is again set at next frame boundary.

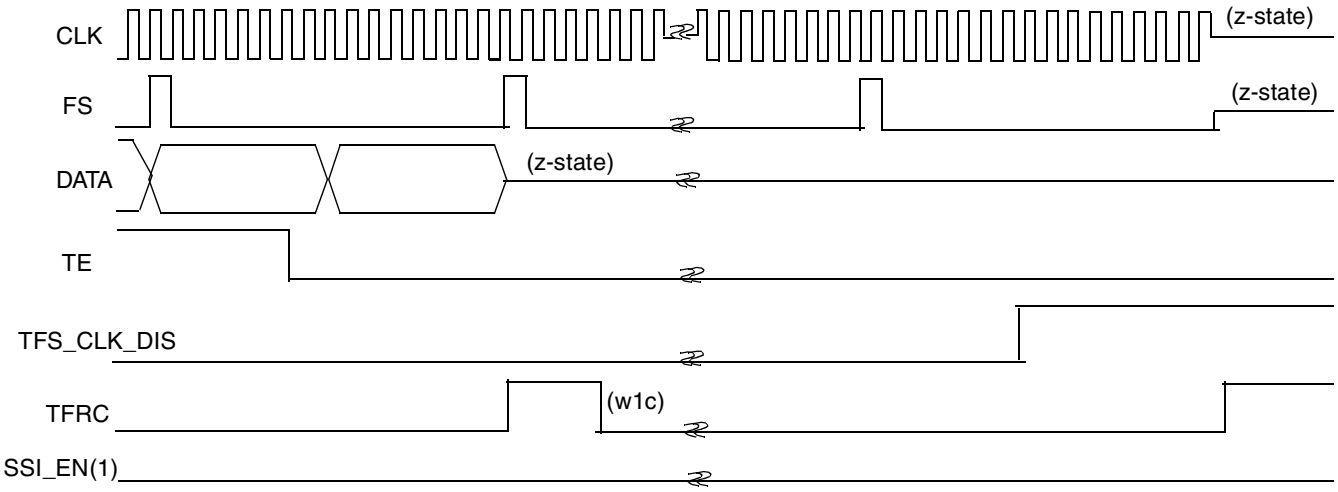


Figure 16-42. TFS_CLK_DIS Assertion in Subsequent Frame after Disabling TE

16.5 Initialization/Application Information

The SSI is affected by the following types of reset:

- Power-on Reset—The Power-on reset is generated by asserting the RESET port. The Power-on reset clears the SSIEN bit in SCR, which disables the SSI. All other status and control bits in the SSI are affected as described in SSI Programming Model in Section 16.3, "SSI Memory Map/Register Definition."
- SSI Reset—The SSI reset is generated when the SSIEN bit in the SCR is cleared. The SSI status bits are preset to the same state produced by the Power-on reset. The SSI control bits are unaffected. The control bits in the SCR are also unaffected. The SSI reset is useful for selective reset of the SSI without changing the present SSI control bits and without affecting the other peripherals.

The correct sequence to initialize the SSI is as follows:

1. Issue a Power-on or SSI reset (SCR[SSIEN]=0).
2. Disable SSI clocks (SOR[CLKOFF], CLKDVDR[SSICKEN]).
3. Set all control bits for configuring the SSI (see Table 16-33 for the list of "SSI Control Bits Requiring SSI to be Disabled Before Change").
4. Enable appropriate interrupts/DMA requests through SIER.
5. Set the SCR[SSIEN] bit (=1) to enable the SSI.
6. Enable SSI clocks (SOR[CLKOFF], CLKDVDR[SSICKEN]), as required.
7. In case of AC97 mode, set the SACNT[AC97EN] bit after programming the SATAG register (if needed, for AC97 Fixed mode).
8. Set SCR[TE/RE] bits.

9. To ensure proper operation of the SSI, use the “Power-on or SSI reset before changing any of the SSI control bits listed in Table 16-33).

NOTE

These control bits should not be changed when SSI is enabled.

Table 16-33. SSI Control Bits Requiring SSI to be Disabled Before Change

Control Register	Bit
CLKDVDR (in Global Utilities)	[24–31]=SSICLK
SCR	[9]=CLK_IST [8]=TCH_EN [6:5]=I2S_MODE [4]=SYN [3]=NET
SIER	[22]=RDMAE [20]=TDMAE
SRCR STCR	[9]=RXBIT0 [9]=TXBIT0 [8]=RFEN1 [8]=TFEN1 [7]=RFEN0 [7]=TFEN0 [6]=RFDIR [6]=TFDIR [5]=RXDIR [5]=TXDIR [4]=RSHFD [4]=TSHFD [3]=RSCKP [3]=TSCKP [2]=RFSI [2]=TFSI [1]=RFSL [1]=TFSL [0]=REFS [0]=TEFS
SRCCR STCCR	[16]=WL3 [15]=WL2 [14]=WL1 [13]=WL0
SACNT	[1]=FV [10:5]=FRDIV

Chapter 17
Global Timer Module

17.1 Introduction

The following sections describe theory of operation of the global timer module (GTM), including a definition of the external signals and the functions they serve. Additionally, the configuration, control, and status registers are described. Note that this chapter describes a single global timer module and that the MPC8610 implements two of these modules.

17.1.1 Overview

The 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 17-1 shows the functional GTM block diagram.

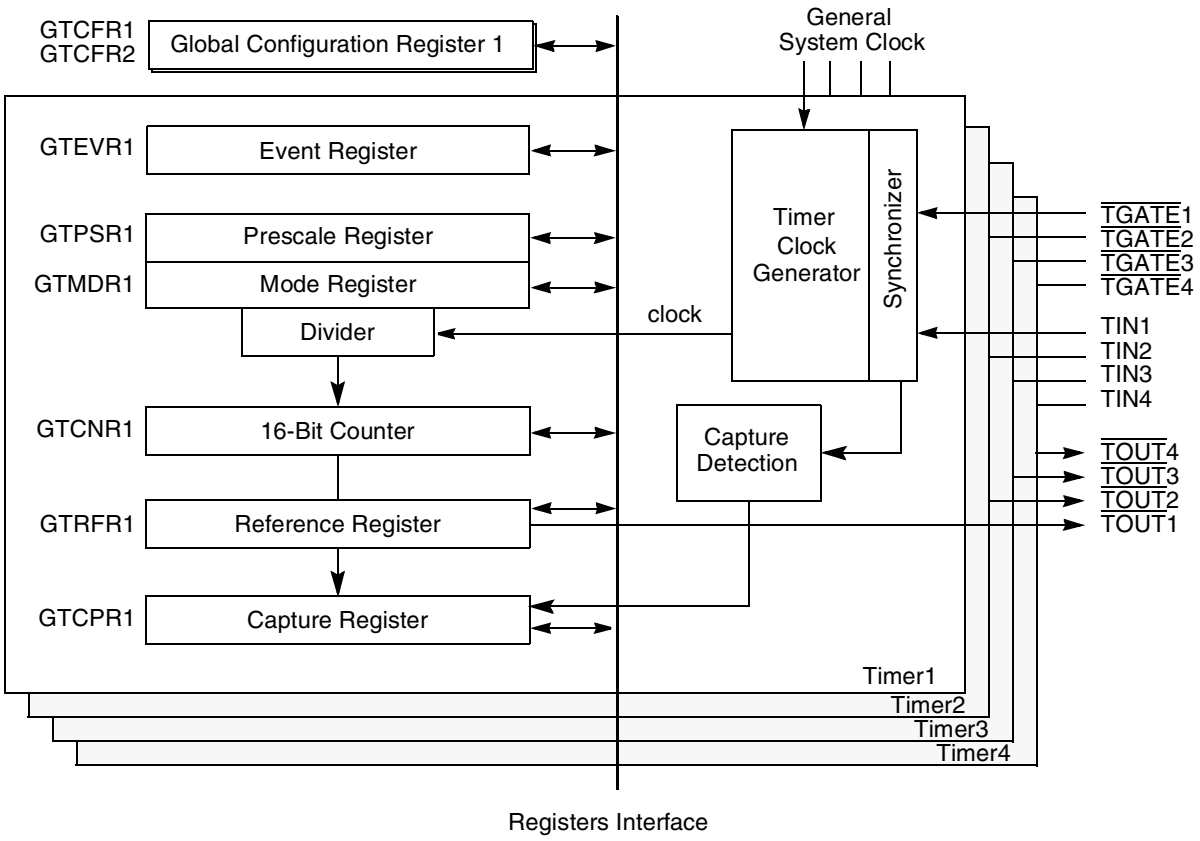


Figure 17-1. Global Timer Module Block Diagram

17.1.2 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) for 16-bit timer
- Maximum period of ~12.8 second (at 333 MHz bus clock) for 32-bit timer
- Maximum period of thousand of year (at 333 MHz bus clock) for 64-bit timer
- 3-nanosecond timer resolution (at 333 MHz bus clock)
- 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

17.1.3 Modes of Operation

The GTM unit can operate in the following modes:

17.1.3.1 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.

17.1.3.2 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

17.1.3.3 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.

17.1.3.4 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.

17.2 External Signal Description

The following sections provide an overview and detailed descriptions of the GTM signals.

NOTE

The GTM*n* signals are multiplexed with GPIO2 signals. See Section 3.2.2, "Global Timer and GPIO2 Signal Multiplexing," for more information. The functionality of these signals is determined by the general-purpose I/O control register (GPIOCR) in the global utilities block. Note that the GPIOCR defaults to GPIO functionality on the GTM signal pins; the GPIOCR must be initialized to the desired GTM signaling for proper operation.

17.2.1 Overview

There are four distinct external input timer capture signals (TIN1, TIN2, TIN3 and TIN4), four distinct external input timer get signals ($\overline{\text{TGATE}}$1, $\overline{\text{TGATE}}$2, $\overline{\text{TGATE}}$3 and $\overline{\text{TGATE}}$4), and four distinct external timer output signals ($\overline{\text{TOUT}}$1, $\overline{\text{TOUT}}$2, $\overline{\text{TOUT}}$3 and $\overline{\text{TOUT}}$4). The GTM interface signals are defined in Table 17-1.

Table 17-1. 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

17.2.2 Detailed Signal Descriptions

Table 17-2 provides detailed descriptions of the external GTM signals.

Table 17-2. GTM External Signals—Detailed Signal Descriptions

Signal	I/O	Description	
TIN*n*	I	Global timer capture control signal. Used to latch the value of the counter when a defined transition of TIN*n* is sensed by the corresponding input capture edge detector.	
		State Meaning	Asserted/Negated —According to the programmed polarity by the corresponding GTMDR*n*[CE]). Each timer has a 16-bit 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. Upon a capture or reference event, the corresponding GTEVR bit is set and a maskable interrupt request is issued to the interrupt controller.
		Timing	Assertion/Negation—Asynchronous to internal bus clock. TIN*n* is internally synchronized to the system bus clock. If TIN*n* meets the asynchronous input setup time, the value of counter is captured after one system bus clock when working with the internal clock.
$\overline{\text{TGATE}}n$	I	Global timer counter gate control signal. Used to gate/restart the counter when a defined transition of $\overline{\text{TGATE}}n$ 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 $\overline{\text{TGATE}}n$ pin is used to enable/disable count. A falling $\overline{\text{TGATE}}n$ pin enables and restarts the count and a rising edge of $\overline{\text{TGATE}}n$ disables the count. In a normal gate mode (GTCFR[GM*n*] = 1), the $\overline{\text{TGATE}}n$ have similar functionality, except the falling edge of $\overline{\text{TGATE}}n$ does not restart the appropriate count value in GTCNR*n*[CNV*n*].
		Timing	Assertion/Negation—Asynchronous to internal bus clock. $\overline{\text{TGATE}}n$ is internally synchronized to the system bus clock. If $\overline{\text{TGATE}}n$ meets the asynchronous input setup time, the counter begins counting after one system bus clock when working with the internal clock.
$\overline{\text{TOUT}}n$	O	Global timer counter output signal. The GTM output a signal on the timer output pin $\overline{\text{TOUT}}n$ 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 $\overline{\text{TOUT}}n$ for one timer input clock cycle as defined by the GTMDR*n*[ICLK*n*] bits (GTMDR*n*[OM*n*] = 1). Thus, $\overline{\text{TOUT}}n$ 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 $\overline{\text{TOUT}}n$ pin (GTMDR*n*[OM*n*] = 0). $\overline{\text{TOUT}}n$ changes occur on the rising edge of the system clock.
		Timing	Assertion/Negation—$\overline{\text{TOUT}}n$ changes occur on the rising edge of the system clock.

17.3 Memory Map/Register Definition

The GTM programmable register map occupies 64 bytes of memory-mapped space. 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.

Table 17-3 shows the memory mapped registers of the GTM 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 CCSRBAR together with the GTM block base address and offset listed in Table 17-3. Undefined 4-byte address spaces within offset 0x00–0xFF are reserved; reading undefined portions of the memory map returns all zeros; writing has no effect.

Table 17-3. GTM Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
GTM1 Registers—Block Base Address 0x1_7000				
0x00	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x00	17.3.1/17-7
0x01–0x03	Reserved	—	—	—
0x04	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x00	17.3.1/17-7
0x05–0x0f	Reserved	—	—	—
0x10	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	17.3.2/17-10
0x12	Timer 2 global timers mode register (GTMDR2)			
0x14	Timer 1 global timers reference register (GTRFR1)	R/W	0x0000	17.3.3/17-11
0x16	Timer 2 global timers reference register (GTRFR2)			
0x18	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	17.3.4/17-12
0x1A	Timer 2 global timers capture register (GTCPR2)			
0x1C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	17.3.5/17-12
0x1E	Timer 2 global timers counter register (GTCNR2)			
0x20	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	17.3.2/17-10
0x22	Timer 4 global timers mode register (GTMDR4)			
0x24	Timer 3 global timers reference register (GTRFR3)	R/W	0x0000	17.3.3/17-11
0x26	Timer 4 global timers reference register (GTRFR4)			
0x28	Timer 3 global timers capture register (GTCPR3)	R	0x0000	17.3.4/17-12
0x2A	Timer 4 global timers capture register (GTCPR4)			
0x2C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	17.3.5/17-12
0x2E	Timer 4 global timers counter register (GTCNR4)			
0x30	Timer 1 global timers event register (GTEVR1)	Special	0x0000	17.3.6/17-13
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	17.3.7/17-13
0x3A	Timer 2 global timers prescale register (GTPSR2)			
0x3C	Timer 3 global timers prescale register (GTPSR3)			
0x3E	Timer 4 global timers prescale register (GTPSR4)			

Table 17-3. GTM Register Address Map (continued)

Offset	Register	Access	Reset Value	Section/ Page
GTM2 Registers—Block Base Address 0x1_7100				
0x00–0x3E	Same as GTM1 registers but offset 0x100			

17.3.1 Global Timers Configuration Registers (GTCFR*n*)

The global timers configuration registers (GTCFR1 and GTCFR2), shown in Figure 17-2 and Figure 17-3, 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. It is allowed to change the mode when clearing GTCFRn[RSTn], but when setting the GTCFRn[RSTn], this bit is the only bit that can be changed.

	0	1	2	3	4	5	6	7
Field	PCAS	BCM	STP2	RST2	GM2	GM1	STP1	RST1
Reset	0000_0000							
R/W	R/W							
Addr	0x00							

Figure 17-2. Global Timers Configuration Register 1 (GTCFR1)

Table 17-4 defines the bit fields of GTCFR1.

Table 17-4. 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

Table 17-4. GTCFR1 Bit Settings (continued)

Bits	Name	Description
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].
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 17-3.

	0	1	2	3	4	5	6	7
Field	PCAS	SCAS	STP4	RST4	GM4	GM3	STP3	RST3
Reset	0000_0000							
R/W	R/W							
Addr	0x04							

Figure 17-3. Global Timers Configuration Register 2 (GTCFR2)

Table 17-5 defines the bit fields of GTCFR2.

Table 17-5. 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 3and 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.

17.3.2 Global Timers Mode Registers (GTMDR1–GTMDR4)

The global timers mode registers (GTMDR1, GTMDR2, GTMDR3 and GTMDR4) are shown in Figure 17-4.

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.

	0 – 7	8 – 9	10	11	12	13 – 14	15
Field	SPS	CE	OM	ORI	FRR	ICLK	GE
Reset	0000_0000_0000_0000						
R/W	R/W						
Addr	0x10(GTMDR1), 0x12(GTMDR2), 0x20(GTMDR3), 0x22(GTMDR4)						

Figure 17-4. Global Timers Mode Registers (GTMDR1—GTMDR4)

Table 17-6 defines the bit fields of GTMDR.

Table 17-6. 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 upon 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 17-6. 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.

17.3.3 Global Timers Reference Registers (GTRFR1–GTRFR4)

Global timers reference registers (GTRFR1, GTRFR2, GTRFR3 and GTRFR4), shown in Figure 17-5, are a 16-bit, memory-mapped, read/write registers containing the 16-bit reference values for the each timer's timeout. The reference value is not reached until GTCNR*n*[CNV] increments to the value in GTRFR*n*[TRV].

	0	15
Field	TRV	
Reset	1111_1111_1111_1111	
R/W	R/W	
Addr	0x14(GTRFR1), 0x16(GTRFR2), 0x24(GTRFR3), 0x26(GTRFR4)	

Figure 17-5. Global Timers Reference Registers (GTRFR1–GTRFR4)

Table 17-7 defines the bit fields of GTRFR.

Table 17-7. 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.

17.3.4 Global Timers Capture Registers (GTCPR1–GTCPR4)

Global timers capture registers (GTCPR1, GTCPR2, GTCPR3 and GTCPR4), shown in Figure 17-6, are used to latch the value of the counters according to GTMDR*n*[CE].

	0	15
Field	LCV	
Reset	000_000_000_000	
R/W	R	
Addr	0x18(GTCPR1), 0x1A(GTCPR2), 0x28(GTCPR3), 0x2A(GTCPR4)	

Figure 17-6. Global Timers Capture Registers (GTCPR1–GTCPR4)

Table 17-8 defines the bit fields of GTCPR*n*.

Table 17-8. GTCPR*n* Bit Settings

Bits	Name	Description
0–15	LCV	Latched counter value. Corresponding timer's 16-bit latched value.

17.3.5 Global Timers Counter Registers (GTCNR1–GTCNR4)

Global timers counter registers (GTCNR1, GTCNR2, GTCNR3 and GTCNR4), shown in Figure 17-7, 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.

	0	15
Field	CNV	
Reset	0000_0000_0000_0000	
R/W	R/W	
Addr	0x1C(GTCNR1), 0x1E(GTCNR2), 0x2C(GTCNR3), 0x2E(GTCNR4)	

Figure 17-7. Global Timers Counter Registers (GTCNR1—GTCNR4)

Table 17-8 defines the bit fields of GTCNR.

Table 17-9. GTCNR Bit Settings

Bits	Name	Description
0–15	CNV	Counter value. Corresponding timer's 16-bit read/write up-counter value.

17.3.6 Global Timers Event Registers (GTEVR1–GTEVR4)

Global timers event registers (GTEVR1, GTEVR2, GTEVR3 and GTEVR4), shown in Figure 17-8, 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.

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.

	0	13	14	15
Field	—		REF	CAP
Reset	0000_0000_0000_0000			
R/W	R/W			
Addr	0x30(GTEVR1), 0x32(GTEVR2), 0x34(GTEVR3), 0x36(GTEVR4)			

Figure 17-8. Global Timers Event Registers (GTEVR1—GTEVR4)

Table 17-10 defines the bit fields of GTEVR*n*.

Table 17-10. 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.

17.3.7 Global Timers Prescale Registers (GTPSR1–GTPSR4)

The global timers prescale registers (GTPSR1, GTPSR2, GTPSR3 and GTPSR4) are shown in Figure 17-9.

Erratic behavior may occur if GTPSR*n* is not initialized before the corresponding GTMDR*n*.

	0	7	8	15
Field	—		PPS	
Reset	0000_0000_0000_0011			
R/W	R/W			
Addr	0x38(GTPSR1), 0x3A(GTPSR2), 0x3C(GTPSR3), 0x3E(GTPSR4)			

Figure 17-9. Global Timers Prescale Registers (GTPSR1–GTPSR4)

Table 17-11 defines the bit fields of GTPSR*n*.

Table 17-11. 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] = 0×00, GTMDR*n*[SPS] = 0×00) to 65,536 (GTPSR*n*[PPS] = 0×FF, GTMDR[SPS] = 0×FF).

17.4 Functional Description

17.4.1 General-Purpose Timer Units

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

- The system clock (*ipg_clock*)
- The system slow go clock (*ipg_clock* internally divided by 16)
- The corresponding TIN*n* signal, usually programmed in the general purpose I/O (GPIO) registers

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—for example, 3 ns for a 333-MHz system clock. In this same example (333 MHz), the maximum period (when the reference value is all ones) for one 16-bit timer is approximately 50 ms.

17.4.2 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.

17.4.3 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 GTGCR.

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.

17.4.4 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 17-10). When working in the none-cascaded mode, the non-cascaded GTRFR, GTCPR, and GTCNR should be referenced with appropriate 16-bit bus cycles.

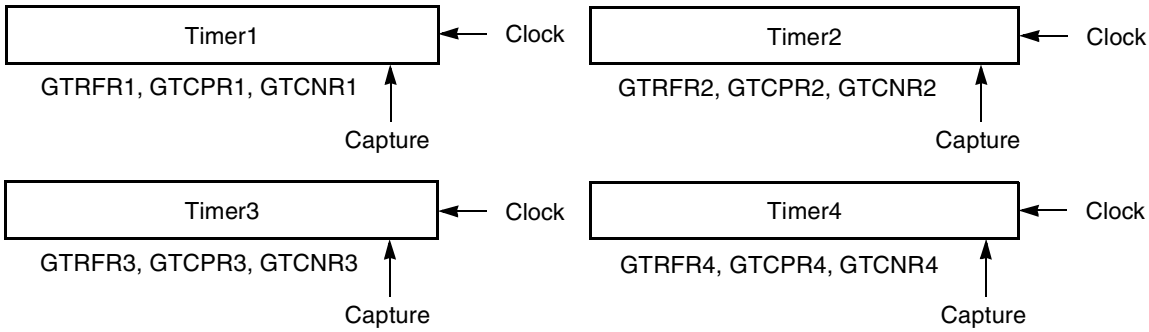


Figure 17-10. Timers Non-Cascaded Mode Block Diagram

- 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 17-11. 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 timer4) 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.

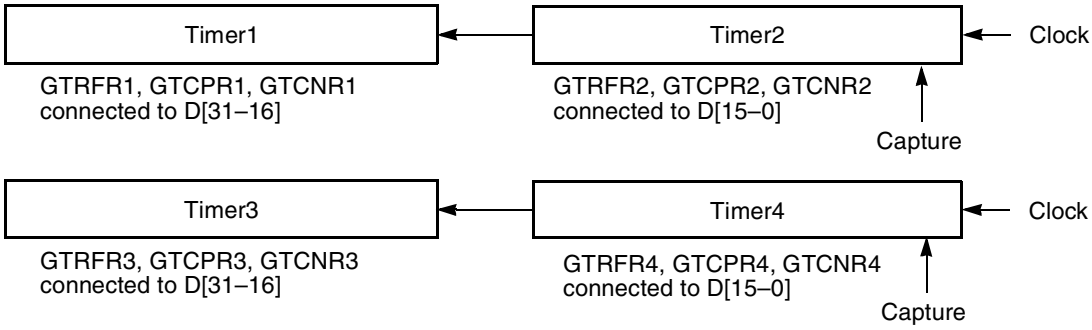


Figure 17-11. Timer Pair-Cascaded Mode Block Diagram

- 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 17-12.

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.

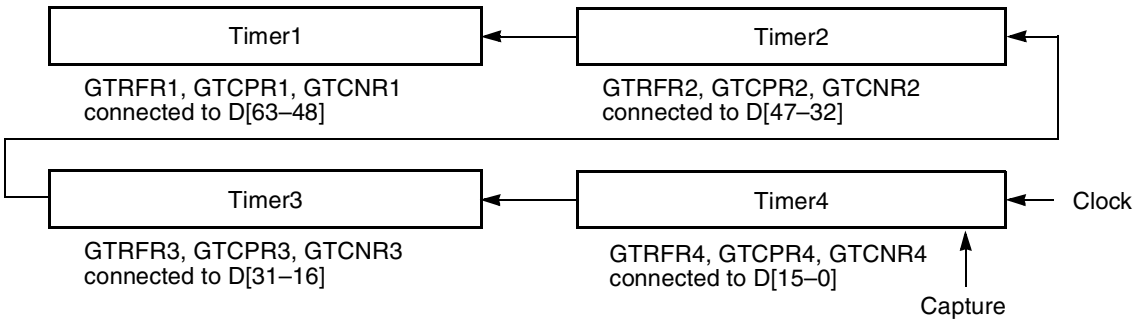


Figure 17-12. Timers Super-Cascaded Mode Block Diagram

17.5 Initialization/Application Information

17.5.1 Programming Guidelines

17.5.1.1 GTM Registers

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 GTGCR and GTPSR is not initialized before the GTMDR. Only GTGCR[RST] can be modified at any time

- Clear GTEVR*n*[REF] and GTEVR*n*[CAP] by writing 1's 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 GTGCR*n*[STP] and to GTGCR*n*[RST] in order to initialize the appropriate timer's operation.

Chapter 18
Watchdog Timer

18.1 Introduction

The following sections describe the theory of operation of the software watchdog timer (WDT) in the MPC8610 device, including a definition of the external signals and the functions they serves. 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.

18.1.1 Overview

The MPC8610 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 18-1 shows a high-level block diagram of the WDT.

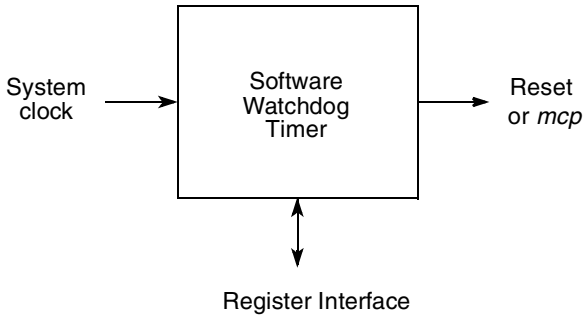


Figure 18-1. Software Watchdog Timer High-Level Block Diagram

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.

18.1.2 Features

The WDT includes the following key features:

- Based on 15-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 platform clock
  ~10.7-sec maximum software time-out delay for 400 MHz platform clock
  ~8.0-sec maximum software time-out delay for 533 MHz platform clock
- Functional and programming compatibility with MPC8260 watchdog timer

18.1.3 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 CRR[SWPR] bit, which controls the divide-by-32,768 of the WDT counter.

18.2 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 block base address.

Table 18-1 shows the WDT memory map.

Table 18-1. WDT Register Address Map

Offset	Register	Access	Reset Value	Section/ Page
Watchdog Timer Registers—Block Base Address 0xE4000				
0x00–0x03	Reserved	—	—	—
0x04	System watchdog control register (SWCRR)	R/W	0x0000_0007	18.2.1/18-3
0x08	System watchdog count register (SWCNR)	R	0x0000_FFFF	18.2.2/18-4
0x0C–0x0D	Reserved	—	—	—
0x0E	System watchdog service register (SWSRR)	R/W	All zeros	18.2.3/18-5

18.2.1 System Watchdog Control Register (SWCRR)

The system watchdog control register (SWCRR), shown in Figure 18-2, 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.

	0	15
Field	SWTC	
Reset	1111_1111_1111_1111	
R/W	R/W	
Addr	0x4	

	16	28	29	30	31
Field	—		SWEN	SWRI	SWPR
Reset	0000_0000_0000_0		S[1]	1	1
R/W	R/W		Read only	R/W	
Addr	0x6				

1 Special: SWCRR[SWEN] reset value depends on cfg_wdt_en configuration signal sampled at reset.

Figure 18-2. System Watchdog Control Register (SWCRR)

Table 18-2 defines the bit fields of SWCRR.

Table 18-2. 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 anytime a new value is loaded into the watchdog counter and also during reset.
16–28	—	Write reserved, read = 0
29	SWEN	Watchdog enable bit. Read-only. This bit is set or cleared depending on the state of the cfg_wdt_en configuration signal sampled at reset. Refer to Section 4.4.3.18, "Watchdog Timer Enable," for more information. 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

Table 18-2. SWCRR Bit Settings (continued)

Bits	Name	Description
30	SWRI	Software watchdog reset/interrupt select bit A WDT timer out causes either a soft 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 the assertion of $\overline{\text{HRESET\_REQ}}$ signal
31	SWPR	Software watchdog counter prescale bit Controls the divide-by-32,768 WDT counter prescaler 0 The WDT counter is not prescaled. 1 The WDT counter clock is prescaled.

18.2.2 System Watchdog Count Register (SWCNR)

The system watchdog count register (SWCNR), shown in Figure 18-3, 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.

	0	15
Field	—	
Reset	0000_0000_0000_0000	
R/W	Read only	
Addr	0x8	

	16	31
Field	SWCN	
Reset	1111_1111_1111_1111	
R/W	Read only	
Addr	0xA	

Figure 18-3. System Watchdog Count Register (SWCNR)

Table 18-3 defines the bit fields of SWCNR.

Table 18-3. 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.

18.2.3 System Watchdog Service Register (SWSRR)

The system watchdog service register (SWSRR) is shown in Figure 18-4. 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.

	0	15
Field	WS	
Reset	0000_0000_0000_0000	
R/W	Write Only (Reads return 0x0000)	
Addr	0xE	

Figure 18-4. System Watchdog Service Register (SWSRR)

Table 18-4 defines the bit fields of SWCNR.

Table 18-4. 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 time-out. SWSRR[WS] can be written at any time, but returns all zeros when read.

18.3 Functional Description

18.3.1 Software Watchdog Timer Unit

The MPC8610 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 0×556C to the system watchdog service register (SWSRR)
- Write 0×AA39 to SWSRR

The service sequence reloads the watchdog timer and the timing process begins again. If a value other than 0×556C or 0×AA39 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 18-5 shows a state diagram for the watchdog timer.

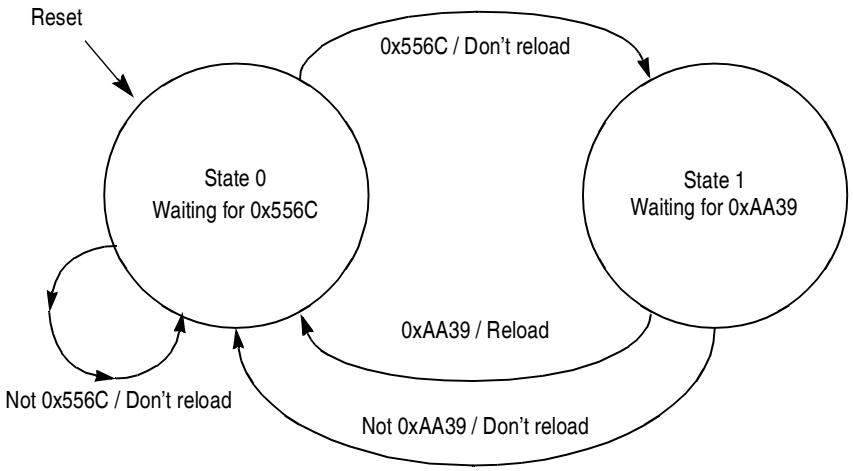


Figure 18-5. Software Watchdog Timer Service State Diagram

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 18-6 shows how to handle this need.

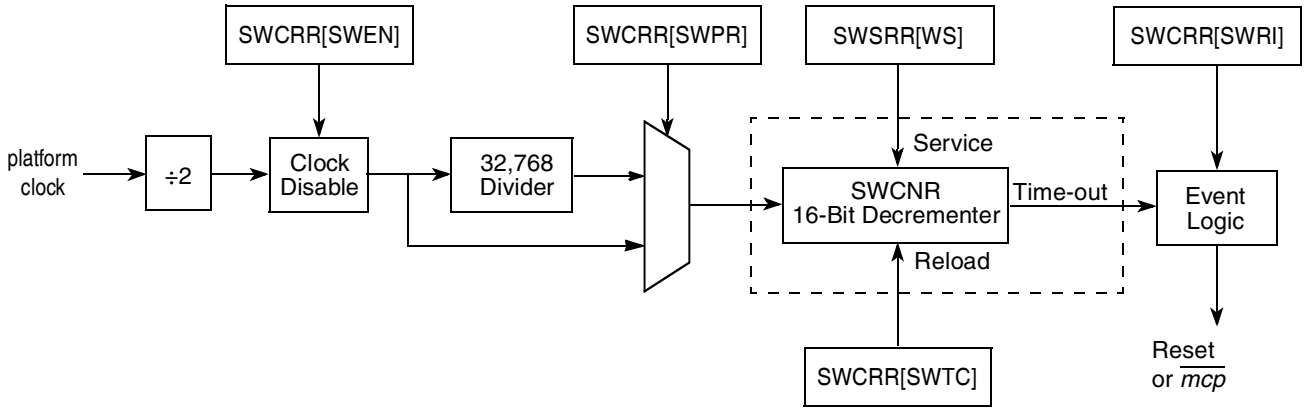


Figure 18-6. Software Watchdog Timer Functional Block Diagram

Figure 18-6 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-32,768 prescaler value is used when needed.

The decrementer begins counting when loaded with a value from SWTC. After the timer reaches 0×0, 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.

18.3.2 Modes of Operation

The WDT unit can operate in the following modes:

- WDT enable/disable mode:

  If the software watchdog timer is not needed for a particular system design, the module can be disabled by pulling the cfg_wdt_en configuration signal down at reset. Refer to Section 4.4.3.18, "Watchdog Timer Enable," for more information. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state.
- 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 soft reset or machine check interrupt to the core.

  — Reset mode (SWCRR[SWRI] = 1).

  Software watchdog timer causes a soft reset (this is the default value after soft 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 CRR[SWPR] bit that controls the divide-by-32,768 of the WDT counter.

  — Prescale mode (CRR[SWPR] = 1)

  The WDT clock is prescaled.

  — Non-prescale mode (CRR[SWPR] = 0)

  The WDT clock is not prescaled.

18.4 Initialization/Application Information

18.4.1 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 initial servicing

  If the software watchdog timer is to be used, the special service sequence, described in Section 18.3.1, “Software Watchdog Timer Unit,” must be executed after system reset and not later than the first WDT time-out (~12.8 sec. for 333 MHz platform clock, ~10.7 sec. for 400 MHz platform clock, or ~8.0 sec. for 533 MHz platform clock).

  Subsequently, periodical WDT servicing should be performed according to the programming guidelines given in Section 18.3.1, “Software Watchdog Timer Unit.”

Chapter 19
DMA Controllers

This chapter describes the DMA controllers offered on this device. This chapter describes a single DMA controller which is instantiated twice on this device. As such, all functionality is identical between the two controllers with the exception of the register offsets, as noted in Table 19-4, and the external signals, as noted in Table 19-3.

19.1 Introduction

The DMA controller transfers blocks of data between the many interface and functional blocks of this device, independent of the e600 core or external hosts.

19.1.1 Block Diagram

Figure 19-1 shows the block diagram of the DMA controller.

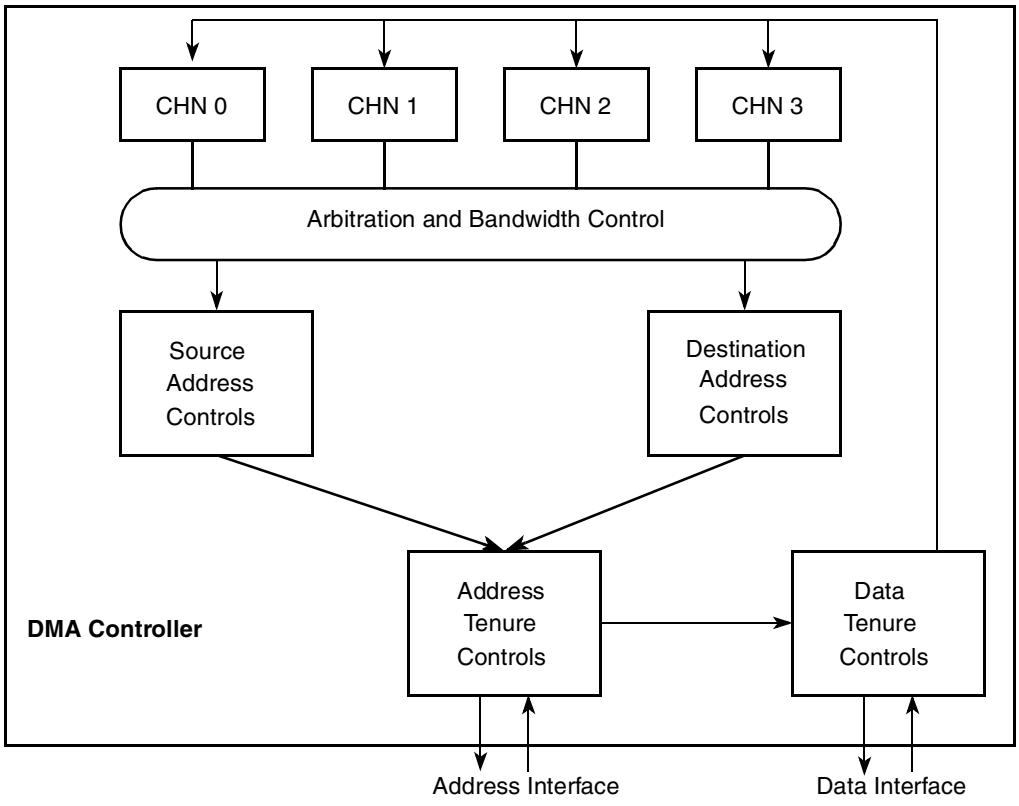


Figure 19-1. DMA Block Diagram

19.1.2 Overview

The DMA controller has four high-speed DMA channels. Both the e600 core and external devices can initiate DMA transfers. The channels are capable of complex data movement and advanced transaction chaining. Figure 19-1 is a high-level block diagram of the DMA controller. Operations such as descriptor fetches and block transfers are initiated by each channel. A channel is selected by the arbitration logic and information is passed to the source and destination control blocks for processing. The source and destination blocks generate read and write requests to the address tenure engine, which manages the DMA master port address interface. After a transaction is accepted by the master port, control is transferred to the data tenure engine that manages the read and write data transfers. A channel remains active in the shared resources for the duration of the data transfer unless the allotted bandwidth per channel is reached.

19.1.3 Features

The DMA controller offers the following features:

- Four high-speed/high-bandwidth channels accessible by local and remote masters
- Basic DMA operation modes (direct, simple chaining)
- Extended DMA operation modes (advanced chaining and stride capability)
- Cascading descriptor chains
- Misaligned transfers
- Programmable bandwidth control between channels
- Up to 256 bytes for DMA sub-block transfers to maximize performance
- Three priority levels supported for source and destination transactions
- Interrupt on error and completed segment, list, or link
- Externally-controlled transfer using $\overline{\text{DMA\_DREQ}}$, $\overline{\text{DMA\_DACK}}$, and $\overline{\text{DMA\_DDONE}}$

19.1.4 Modes of Operation

The DMA block has two modes of operation: basic and extended. Basic mode is the DMA legacy mode, which does not support advanced features. Extended mode supports advanced features such as striding and flexible descriptor structures.

These two basic modes allow users to initiate and end DMA transfers in various ways. Table 19-1 summarizes the relationship between the modes and the following features:

- Direct mode—No descriptors are involved. Software must initialize the required fields as described in Table 19-1 before starting a transfer.
- Chaining mode—Software must initialize descriptors in memory and the required fields as described in Table 19-1 before starting a transfer.
- Single-write start mode—The DMA process can be started using a single-write command to either the descriptor address register in one of the chaining modes or the source/destination address registers in one of the direct modes.
- External control capability—This allows an external agent to start, pause, and check the status of a DMA transfer that has already been initialized.

- Channel continue capability—The channel continue capability allows software the flexibility of having the DMA controller start with descriptors that have already been programmed while software continues to build more descriptors in memory.
- Channel abort capability—The software can abort a previously initiated transfer by setting the bit MR*n*[CA]. The DMA controller terminates all outstanding transfers initiated by the channel without generating any errors before entering an idle state.

Table 19-1. Relationship of Modes and Features

Mode	Mode with One Additional Feature	Mode with Two Additional Features
B (Basic)	BD (basic direct)	BDS (BD single-write start)
		BDE (BD external control)
	BC (basic chaining)	BCE (BC external control)
		BCS (BC single-write start)
Ext (Extended)	ExtD (extended direct)	ExtDS (ExtD single-write start)
		ExtDE (ExtD external control)
	ExtC (extended chaining)	ExtCE (ExtC external control)
		ExtCS (ExtC single-write start)

Table 19-2 describes bit settings required for each DMA mode of operation.

Table 19-2. DMA Mode Bit Settings

Modes with Features	MR*n*[XFE]	MR*n*[CTM]	MR*n*[SRW]	MR*n*[CDSM/SWSM]	MR*n*[EMS_EN]
Basic Direct Modes					
Basic direct	0	1	0	0	0
Basic direct external control	0	1	0	0	1
Basic direct single-write start	0	1	1	1 or 0	0
Basic Chaining Modes					
Basic chaining	0	0	Reserved	0	0
Basic chaining external control	0	0	Reserved	0	1
Basic chaining single-write start	0	0	Reserved	1	0
Extended Direct Modes					
Extended direct	1	1	0	0	0
Extended direct external control	1	1	0	0	1
Extended direct single-write start	1	1	1	1 or 0	0

Table 19-2. DMA Mode Bit Settings (continued)

Modes with Features	MR*n*[XFE]	MR*n*[CTM]	MR*n*[SRW]	MR*n*[CDSM/SWSM]	MR*n*[EMS_EN]
Extended Chaining Modes					
Extended chaining	1	0	Reserved	0	0
Extended chaining external control	1	0	Reserved	0	1
Extended chaining single-write start	1	0	Reserved	1	0

Refer to Section 19.4, “Functional Description,” for details on these modes.

Figure 19-2 shows the general DMA operational flow chart.

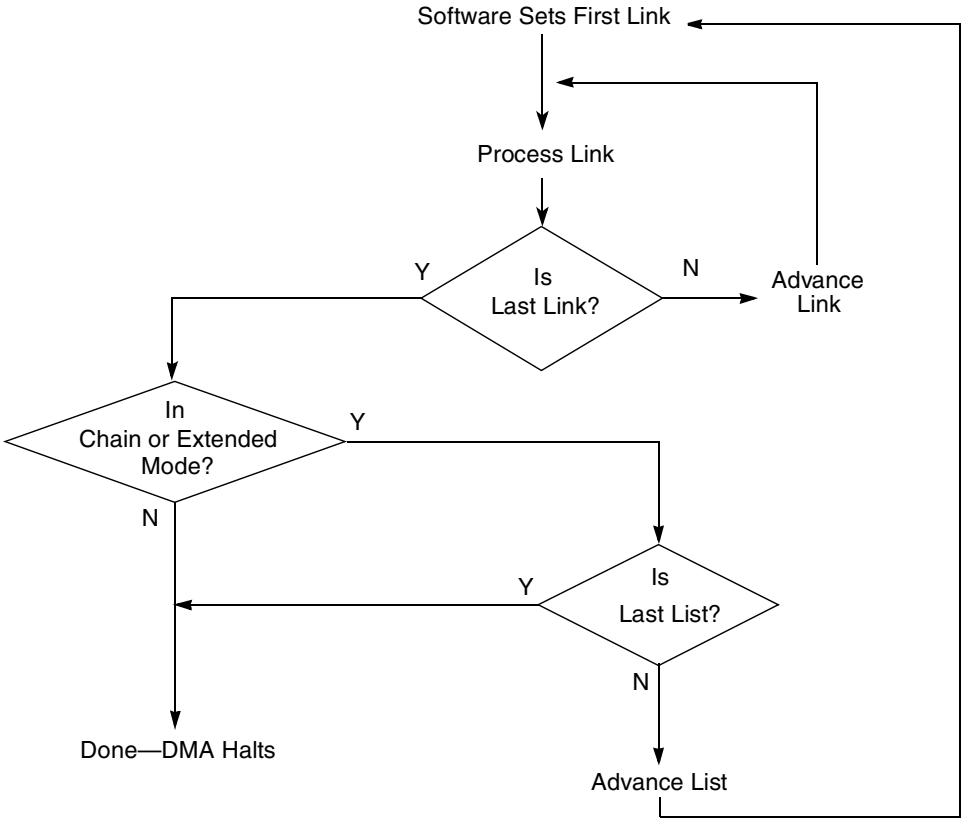


Figure 19-2. DMA Operational Flow Chart

19.2 External Signal Description

This section describes the DMA signals.

19.2.1 Signal Overview

Figure 19-3 summarizes the DMA controller signals. Note that DMA channels 2 and 3 are multiplexed with local bus and interrupt signals as follows:

- $\overline{\text{DMA\_DREQ[2]}}/\overline{\text{LCS[5]}}$

- DMA_DACK[2]/LCS[6]
- DMA_DDONE[2]/LCS[7]
- DMA_DREQ[3]/IRQ[9]
- DMA_DACK[3]/IRQ[10]
- DMA_DDONE[3]/IRQ[11]

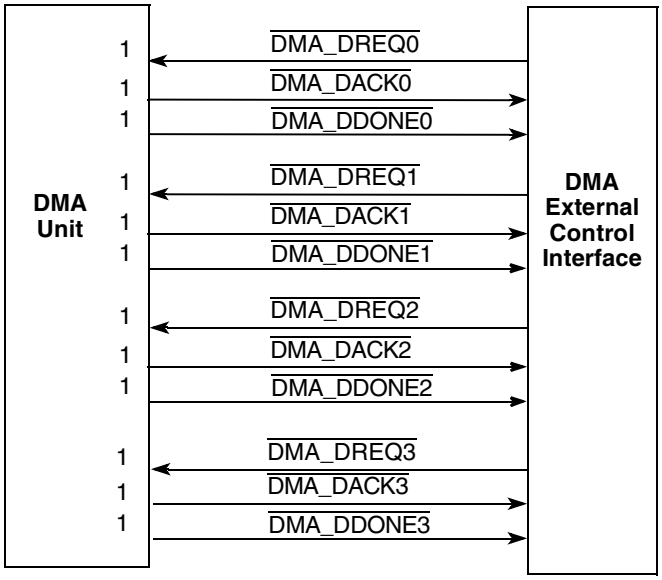


Figure 19-3. DMA Signal Summary

Note that the three DMA signals for DMA channel 3 are multiplexed with the IRQ9–11 signals on the device. These functions are mutually exclusive and the active function is specified in the PMUXCR register of the global utilities block as described in Section 23.4.1.9, "Alternate Function Signal Multiplex Control Register (PMUXCR)."

19.2.2 Detailed Signal Descriptions

Table 19-3 describes the DMA signals.

Table 19-3. DMA Signals—Detailed Signal Descriptions

Signal	I/O	Description	
DMA_DREQ*n* DMA request	I	DMA request. Indicates the start of a DMA transfer or a restart from a paused request. Assertion of DMA_DREQ*n* causes MR*n*[CS] to be set, thereby activating the corresponding DMA channel.	
		State Meaning	Asserted—Assertion of DMA_DREQ*n* while DMA_DACK*n* is negated causes a new transfer to start OR resumes a paused transfer if the EMP_EN bit is set. Assertion while DMA_DACK*n* is asserted results in an illegal condition. Negated—Negation while DMA_DACK*n* is asserted has no effect. Negation before the assertion of DMA_DACK*n* results in an illegal condition.
		Timing	Assertion—Can be asserted asynchronously Negation— Must remain asserted at least until the assertion of the corresponding DMA_DACK*n*

Table 19-3. DMA Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
DMA_DACK*n*	O	DMA acknowledge. Indicates that a DMA transfer is currently in progress	
		State Meaning	Asserted—Indicates that a DMA transfer is currently in progress. Asserted after the assertion of DMA_DREQ*n* to indicate the start of a transfer Negated—Negated after finishing a complete transfer or after entering a paused state if MR*n*[EMP_EN] is set
		Timing	Assertion—Asynchronous assertion; asserted for more than three system clocks Negation—Asynchronous negation; negated for more than three system clocks
DMA_DDONE*n*	O	DMA done. Indicates that a DMA transfer is complete	
		State Meaning	Asserted—Indicates transfer completion. SRn[CB] is clear. Note, however, that write data may still be queued at the target interface or in the process of transfer on an external interface. Negated—Indicates that the current transfer is in process
		Timing	Assertion—Always asserts asynchronously after the negation of the final DMA_DACK*n* to indicate completion of a transfer. For a paused transfer, DMA_DDONE*n* is asserted asynchronously after the negation of the final DMA_DACK*n*. Negation—Negated asynchronously after the assertion of DMA_DREQ*n* for the next transfer

19.3 Memory Map/Register Definition

This section provides a detailed description of all accessible DMA memory and registers. The descriptions include individual bit level descriptions and reset states of each register. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved.

Table 19-4 lists the DMA registers and their offsets. Note that the full register address is comprised of the programmable CCSRBAR together with the fixed DMA block base address and offset listed in Table 19-4.

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 19-4. DMA Register Summary

Offset	Register	Access	Reset	Section/Page
DMA Controller 1 Block Base Address: 0x2_1000				
0x100	MR0—DMA 0 mode register	R/W	All zeros	19.3.1.1/19-9
0x104	SR0—DMA 0 status register	Mixed	All zeros	19.3.1.2/19-12
0x108	ECLNDAR0—DMA 0 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13

Table 19-4. DMA Register Summary (continued)

Offset	Register	Access	Reset	Section/Page
0x10C	CLNDAR0—DMA 0 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x110	SATR0—DMA 0 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x114	SAR0—DMA 0 source address register	R/W	All zeros	19.3.1.5/19-16
0x118	DATR0—DMA 0 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x11C	DAR0—DMA 0 destination address register	R/W	All zeros	19.3.1.7/19-18
0x120	BCR0—DMA 0 byte count register	R/W	All zeros	19.3.1.8/19-19
0x124	ENLNDAR0—DMA 0 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x128	NLNDAR0—DMA 0 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x130	ECLSDAR0—DMA 0 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x134	CLSDAR0—DMA 0 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x138	ENLSDAR0—DMA 0 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x13C	NLSDAR0—DMA 0 next list descriptor address register	Mixed	All zeros	19.3.1.11/19-22
0x140	SSR0—DMA 0 source stride register	R/W	All zeros	19.3.1.12/19-23
0x144	DSR0—DMA 0 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x148–0x17C	Reserved	—	—	—
0x180	MR1—DMA 1 mode register	R/W	All zeros	19.3.1.1/19-9
0x184	SR1—DMA 1 status register	Mixed	All zeros	19.3.1.2/19-12
0x188	ECLNDAR1—DMA 1 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x18C	CLNDAR1—DMA 1 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x190	SATR1—DMA 1 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x194	SAR1—DMA 1 source address register	R/W	All zeros	19.3.1.5/19-16
0x198	DATR1—DMA 1 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x19C	DAR1—DMA 1 destination address register	R/W	All zeros	19.3.1.7/19-18
0x1A0	BCR1—DMA 1 byte count register	R/W	All zeros	19.3.1.8/19-19
0x1A4	ENLNDAR1—DMA 1 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x1A8	NLNDAR1—DMA 1 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x1B0	ECLSDAR1—DMA 1 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x1B4	CLSDAR1—DMA 1 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x1B8	ENLSDAR1—DMA 1 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x1BC	NLSDAR1—DMA 1 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22

Table 19-4. DMA Register Summary (continued)

Offset	Register	Access	Reset	Section/Page
0x1C0	SSR1—DMA 1 source stride register	R/W	All zeros	19.3.1.12/19-23
0x1C4	DSR1—DMA 1 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x1C8–0x1FC	Reserved	—	—	—
0x200	MR2—DMA 2 mode register	R/W	All zeros	19.3.1.1/19-9
0x204	SR2—DMA 2 status register	Mixed	All zeros	19.3.1.2/19-12
0x208	ECLNDAR2—DMA 2 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x20C	CLNDAR2—DMA 2 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x210	SATR2—DMA 2 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x214	SAR2—DMA 2 source address register	R/W	All zeros	19.3.1.5/19-16
0x218	DATR2—DMA 2 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x21C	DAR2—DMA 2 destination address register	R/W	All zeros	19.3.1.7/19-18
0x220	BCR2—DMA 2 byte count register	R/W	All zeros	19.3.1.8/19-19
0x224	ENLNDAR2—DMA 2 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x228	NLNDAR2—DMA 2 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x230	ECLSDAR2—DMA 2 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x234	CLSDAR2—DMA 2 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x238	ENLSDAR2—DMA 2 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x23C	NLSDAR2—DMA 2 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22
0x240	SSR2—DMA 2 source stride register	R/W	All zeros	19.3.1.12/19-23
0x244	DSR2—DMA 2 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x248–0x27C	Reserved	—	—	—
0x280	MR3—DMA 3 mode register	R/W	All zeros	19.3.1.1/19-9
0x284	SR3—DMA 3 status register	Mixed	All zeros	19.3.1.2/19-12
0x288	ECLNDAR3—DMA 3 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x28C	CLNDAR3—DMA 3 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x290	SATR3—DMA 3 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x294	SAR3—DMA 3 source address register	R/W	All zeros	19.3.1.5/19-16
0x298	DATR3—DMA 3 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x29C	DAR3—DMA 3 destination address register	R/W	All zeros	19.3.1.7/19-18
0x2A0	BCR3—DMA 3 byte count register	R/W	All zeros	19.3.1.8/19-19

Table 19-4. DMA Register Summary (continued)

Offset	Register	Access	Reset	Section/Page
0x2A4	ENLNDAR3—DMA 3 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x2A8	NLNDAR3—DMA 3 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x2B0	ECLSDAR3—DMA 3 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x2B4	CLSDAR3—DMA 3 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x2B8	ENLSDAR3—DMA 3 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x2BC	NLSDAR3—DMA 3 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22
0x2C0	SSR3—DMA 3 source stride register	R/W	All zeros	19.3.1.12/19-23
0x2C4	DSR3—DMA 3 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x2C8–0x2FC	Reserved	—	—	—
0x300	DGSR—DMA general status register	R	All zeros	19.3.1.14/19-24
DMA Controller 2 Block Base Address:0x0_C000				
0x000–0xFFC	DMA 2 registers **Note:** All registers defined for DMA controller 1 are also defined for DMA controller 2; the offsets of DMA controller 2 registers are the same except they have a different block base address.			

19.3.1 DMA Register Descriptions

The following sections describe the DMA registers. The majority of these registers are channel-specific and can be identified by one of the four offsets that describe the register.

19.3.1.1 Mode Registers (MR*n*)

The mode register allows software to start a DMA transfer and to control various DMA transfer characteristics. Figure 19-4 describes the MR*n*.

Offset 0x100, 0x180, 0x200, 0x280 — Access: Read/Write

	0–3	4–7	8–9	10	11–12	13	14–15
R/W	—	BWC	—	EMP_EN	—	EMS_EN	DAHTS
Reset	All zeros						

	16–17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R/W	SAHTS	DAHE	SAHE	—	SRW	EOSIE	EOLNIE	EOLSIE	EIE	XFE	CDSM/SWSM	CA	CTM	CC	CS
Reset	All zeros														

Figure 19-4. DMA Mode Registers (MR*n*)

Table 19-5 describes the MR*n* fields.

Table 19-5. MR*n* Field Descriptions

Bits	Name	Description
0–3	—	Reserved
4–7	BWC	Bandwidth/pause control. If multiple channels are executing transfers concurrently the value of MR*n*[BWC] determines how many bytes a given channel is allowed to transfer before the DMA engine pauses the current channel and switches to the next channel. If only one channel is executing transfers the value of MR*n*[BWC] dictates how many bytes are allowed to transfer before pausing the channel, after which a new assertion of $\overline{\text{DREQ}}$ resumes channel operation. 0000 1 byte 0001 2 bytes 0010 4 bytes 0011 8 bytes 0100 16 bytes 0101 32 bytes 0110 64 bytes 0111 128 bytes 1000 256 bytes 1001 512 bytes 1010 1024 bytes 1011–1110 Reserved 1111 Disable bandwidth sharing to allow uninterrupted transfers from each channel.
8–9	—	Reserved
10	EMP_EN	External master pause enable. Valid only if MR*n*[EMS_EN] is set. 0 Disable the external master pause feature. 1 Enable the external master pause feature. Channel is paused as described by MR*n*[BWC].
11–12	—	Reserved
13	EMS_EN	External master start enable. This bit does not apply to single-write start modes (direct or chaining). 0 Disable the channel from being started by an external DMA start pin. 1 Enable the channel to be started by an external DMA start pin, which sets MR*n*[CS].
14–15	DAHTS	Destination address hold transfer size. Indicates the transfer size used for each transaction while MR*n*[DAHE] is set. The byte count register must be in multiples of the size and the destination address register must be aligned based on the size. The transfer size assigned to MR*n*[DAHTS] must be equal to or smaller than that assigned to MR*n*[BWC] to avoid undefined behavior. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
16–17	SAHTS	Source address hold transfer size. Indicates the transfer size used for each transaction while MR*n*[SAHE] is set. The byte count register must be in multiples of the size and the source address register must be aligned based on the size. The transfer size assigned to MR*n*[SAHTS] must be equal to or smaller than that assigned to MR*n*[BWC] to avoid undefined behavior. 00 1 byte 01 2 bytes 10 4 bytes 11 8 bytes
18	DAHE	Destination address hold enable 0 Disable destination address hold 1 Enable the DMA controller to hold the destination address of a transfer to the size specified by MR*n*[DAHTS]. Hardware only supports aligned transfers for this feature.

Table 19-5. MR*n* Field Descriptions (continued)

Bits	Name	Description
19	SAHE	Source address hold enable 0 Disable source address hold 1 Enable the DMA controller to hold the source address of a transfer to the size specified by MR*n*[SAHTS]. Hardware only supports aligned transfers for this feature.
20	—	Reserved
21	SRW	Single register write (Direct mode only; reserved for chaining mode.) 0 Normal operation 1 Enable a write to the source address register to simultaneously set MR*n*[CS], starting a DMA transfer, when MR*n*[CDSM/SWSM] is also set. Setting this bit and clearing CDSM/SWSM causes a write to the destination address register to simultaneously set MR*n*[CS], starting a DMA transfer.
22	EOSIE	End-of-segments interrupt enable 0 Do not generate an interrupt at the completion of a data transfer. CLNDAR*n*[EOSIE] overrides this bit on a link descriptor basis. 1 Generate an interrupt at the completion of a data transfer (That is, SR*n*[EOSI] is set). This bit overrides the CLNDAR*n*[EOSIE].
23	EOLNIE	End-of-links interrupt enable 0 Do not generate an interrupt at the completion of a list of DMA transfers. 1 Generate an interrupt at the completion of a list of DMA transfers (That is, NLNDAR*n*[EOLND] is set).
24	EOLSIE	End-of-lists interrupt enable 0 Do not generate an interrupt at the completion of all DMA transfers. 1 Generate an interrupt at the completion of all DMA transfers (That is, NLNDAR*n*[EOLND] and NLSDAR*n*[EOLSD] are set).
25	EIE	Error interrupt enable 0 Do not generate an interrupt if a programming or transfer error is detected. 1 Generate an interrupt if a programming or transfer error is detected.
26	XFE	Extended features enable 0 Disable the new chaining features. 1 Enable the new chaining features.
27	CDSM/ SWSM	In chaining mode, current descriptor start mode/single-write start mode is as follows: • In basic mode (MR*n*[XFE] is cleared), setting this bit causes a write to the current link descriptor address register to simultaneously set MR*n*[CS], starting a DMA transfer. • In extended chaining mode (MR*n*[XFE] is set), setting this bit causes a write to the current list descriptor address register to simultaneously set MR*n*[CS], starting a DMA transfer. In direct mode, setting this bit and MR*n*[SRW] causes a write to the source address register to simultaneously set MR*n*[CS], starting a DMA transfer. Clearing this bit and setting MR*n*[SRW] causes a write to the destination address register to simultaneously set MR*n*[CS], starting a DMA transfer. This bit must be cleared when MR*n*[SRW] is cleared.
28	CA	Channel abort 0 No effect 1 Cause the current transfer to be aborted and SR*n*[CB] to be cleared if the channel is busy. The channel remains in the idle state until a new transfer is programmed.
29	CTM	Channel transfer mode 0 Configure the channel in chaining mode. 1 Configure the channel into direct mode. This means that software is responsible for placing all the required parameters into necessary registers to start the DMA process.

Table 19-5. MR*n* Field Descriptions (continued)

Bits	Name	Description
30	CC	Channel continue. This bit applies only to chaining mode and is cleared by hardware after the first descriptor read when continuing a transfer. This bit is reserved for external master mode. 0 No effect 1 The DMA transfer restarts the transferring process starting at the current descriptor address.
31	CS	Channel start. This bit is also automatically set by hardware during single-write start mode and external master start enable mode. Note that in external control mode, deasserting $\overline{\text{DMA\_DREQ}}$ does NOT clear this bit. 0 Halt the DMA process if channel is busy (SR*n*[CB] is set). No effect if the channel is not busy. 1 Start the DMA process if channel is not busy (CB is cleared). If the channel was halted (CS = 0 and CB = 1), the transfer continues from the point at which it was halted.

19.3.1.2 Status Registers (SR*n*)

The status registers, shown in Figure 19-5, report various DMA conditions during and after a DMA transfer.

Offset 0x104, 0x184, 0x204, 0x284 — Access: Mixed

	0–23	24	25	26	27	28	29	30	31
R	—	TE	—	CH	PE	EOLNI	CB	EOSI	EOLSI
W		w1c			w1c	w1c		w1c	w1c
Reset	All zeros								

Figure 19-5. Status Registers (SR*n*)

Table 19-6 describes the bits of the SR*n*.

Table 19-6. SR*n* Field Descriptions

Bits	Name	Description
0–23	—	Reserved
24	TE	Transfer error (Bit reset, write 1 to clear) 0 No error condition during the DMA transfer 1 Error condition during the DMA transfer. See Section 19.4.3, “DMA Errors,” for additional information.
25	—	Reserved
26	CH	Channel halted. Cleared automatically by hardware if MR*n*[CS] is set again for resuming a halted transfer 0 Channel is not halted. If software attempts to halt an idle channel (SR*n*[CB] is cleared), this bit remains 0. 1 DMA transfer was successfully halted by software and can be resumed.
27	PE	Programming error (bit reset, write 1 to clear) 0 No programming error detected 1 A programming error is detected that prevents the DMA transfer from occurring.
28	EOLNI	End-of-links interrupt. After transferring the last block of data in the last link descriptor, if MR*n*[EOLNIE] is set, then this bit is set and an interrupt is generated. (Bit reset, write 1 to clear)

Table 19-6. SR*n* Field Descriptions (continued)

Bits	Name	Description
29	CB	Channel busy 0 DMA transfer is finished, an error occurred, or a channel abort occurred. 1 DMA transfer is currently in progress.
30	EOSI	End-of-segment interrupt. In chaining mode, after finishing a data transfer, if MR*n*[EOSIE] is set or if CLNDAR*n*[EOSIE] is set, this bit gets set and an interrupt is generated. In direct mode, if MRn[EOSIE] is set, this bit gets set and an interrupt is generated. (Bit reset, write 1 to clear)
31	EOLSI	End-of-list interrupt. After transferring the last block of data in the last list descriptor, if MR*n*[EOLSIE] is set, then this bit is set and an interrupt is generated. (Bit reset, write 1 to clear)

19.3.1.3 Current Link Descriptor Address Registers (CLNDAR*n* and ECLNDAR*n*)

Current link descriptor address registers contain the address of the current link descriptor. In basic chaining mode, shown in Figure 19-6, software must initialize these registers to point to the first link descriptors in memory.

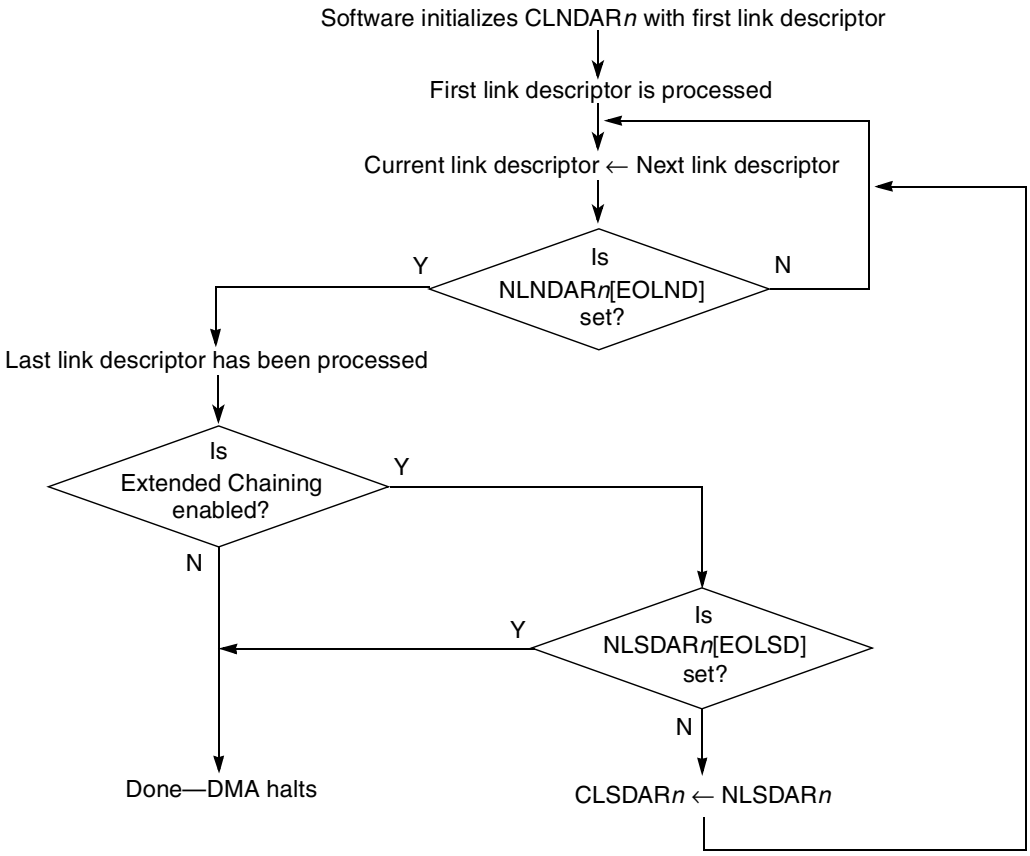


Figure 19-6. Basic Chaining Mode Flow Chart

After the current descriptor is processed, the current link descriptor address register is loaded from the next link descriptor address registers and NLNDAR*n*[EOLND] in the next link descriptor address register is

examined. If EOLND is zero, the DMA controller reads in the new current link descriptor for processing. If EOLND is set, the last descriptor of the list was just completed. If extended chaining mode is not enabled, all DMA transfers are complete and the DMA controller halts.

If extended chaining mode is enabled, the DMA controller examines the state of NLSDAR*n*[EOLSD] in the next list descriptor address register. If EOLSD is clear, the controller loads the contents of the next list descriptor address register into the current list descriptor address register and reads the new list descriptor from memory. If EOLSD is set, all DMA transfers are complete and the DMA controller halts.

Figure 19-7 shows ECLNDAR*n*.

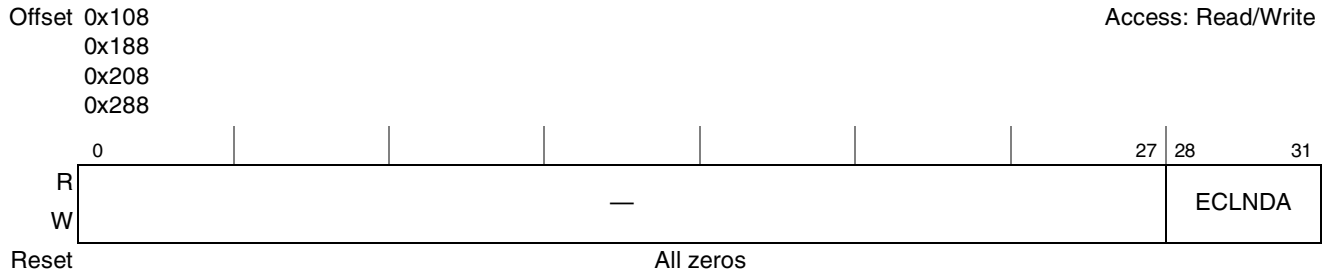


Figure 19-7. Extended Current Link Descriptor Address Registers (ECLNDAR*n*)

Table 19-7 describes the fields of the ECLNDAR*n*.

Table 19-7. ECLNDAR*n* Field Descriptions

Bit	Name	Description
0–27	—	Reserved
28–31	ECLNDA	Current link descriptor extended address (upper 4 bits of 36-bit address)

Figure 19-8 shows CLNDAR*n*.

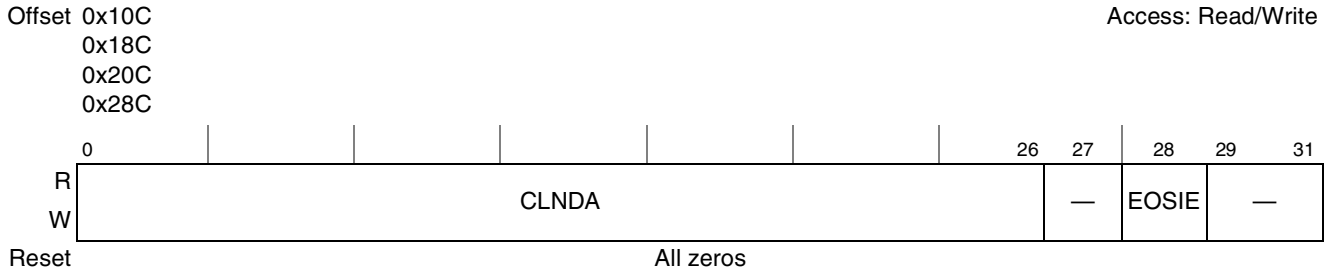


Figure 19-8. Current Link Descriptor Address Registers (CLNDAR*n*)

Table 19-8 describes the fields of the CLNDAR*n*.

Table 19-8. CLNDAR*n* Field Descriptions

Bits	Name	Description
0–26	CLNDA	Current link descriptor address. Contains the current descriptor address of the buffer descriptor in memory. The descriptor must be aligned to a 32-byte boundary. (This is the lower portion of the 36-bit address formed by CLNDAR*n*[CLNDA] and ECLNDAR*n*[ECLNDA].)
27	—	Reserved
28	EOSIE	End-of-segment interrupt enable 0 Do not generate an interrupt upon completion of the current DMA transfer for the current link descriptor. 1 Generate an interrupt upon completion of the current DMA transfer for the current link descriptor.
29–31	—	Reserved

19.3.1.4 Source Attributes Registers (SATR*n*)

The source attributes registers, shown in Figure 19-9, contain the transaction attributes to be used for the DMA operation. Stride mode is enabled by setting SATR*n*[SSME].

Offset 0x110
0x190
0x210
0x290

Access: Read/Write

	0	1	2	3	4	5	6	7	8	11	12	15	16	27	28	31
R W	—		SPBATMU	—	STFLOWLVL		SPCIORDER	SSME	STRANSINT		SREADTTYPE		—		ESAD	
Reset	All zeros															

Figure 19-9. Source Attributes Registers (SATR*n*)

Table 19-9 describes the fields of the SATR*n*.

Table 19-9. SATR*n* Field Descriptions

Bits	Name	Description
0–6	—	Reserved
7	SSME	Source stride mode enable 0 Stride mode disabled 1 Stride mode enabled Ignored in basic mode (MR*n*[XFE] is cleared). Striding on the source address can be accomplished by enabling SATR*n*[SSME] and setting the desired stride size and distance in the SSR*n*.
8–11	—	Reserved

Table 19-9. SATR*n* Field Descriptions (continued)

Bits	Name	Description
12–15	SREADTTYPE	DMA source transaction type. Reserved values result in a programming error being detected and logged in SR[PE]. Transaction type to run on local address space. 0000–0011 Reserved 0100 Read, do not snoop local processor 0101 Read, snoop local processor
16–27	—	Reserved
28–31	ESAD	Extended source address. ESAD represents the four high-order bits of the 36-bit source address.

19.3.1.5 Source Address Registers (SAR*n*)

The source address registers, shown in Figure 19-10, contain the address from which the DMA controller reads data. In direct mode, if MR*n*[CDSM/SWSM] and MR*n*[SRW] are set, a write to this register simultaneously sets MR*n*[CS], starting a DMA transfer. Software must ensure that this is a valid address.

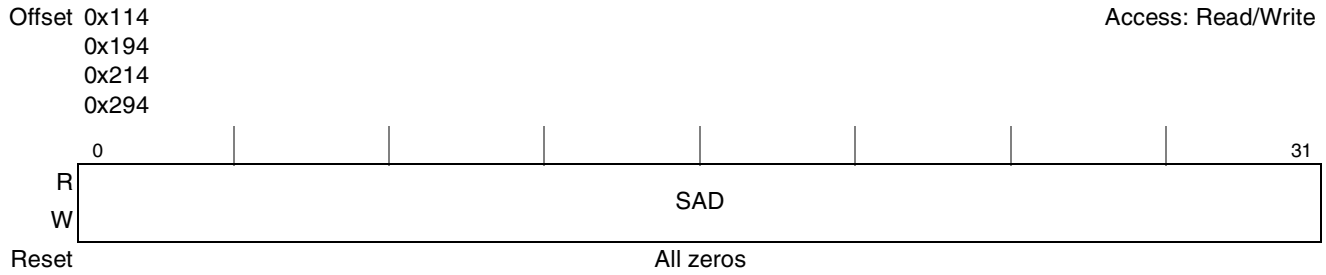


Figure 19-10. Source Address Registers (SAR*n*)

Table 19-10 describes the field of the SAR*n*.

Table 19-10. SAR*n* Field Descriptions

Bits	Name	Description
0–31	SAD	Source address. This register contains the low-order bits of the 36-bit source address of the DMA transfer. The contents are updated after every DMA write operation unless the final stride of a striding operation is less than the stride size, in which case it remains equal to the address from which the last stride began.

19.3.1.6 Destination Attributes Registers (DATR*n*)

The destination attributes registers, shown in Figure 19-11, contain the transaction attributes for the DMA operation. Stride mode is enabled by setting DATR*n*[DSME].

Offset 0x118
0x198
0x218
0x298

Access: Read/Write

	0	1	2	3	4	5	6	7	8	11	12	15	16	27	28	31
R W	—		DPBATMU	—	DTFLOWLVL		DPCIORDER	DSME	DTRANSINT		DWRITETTYPE		—		EDAD	
Reset	All zeros															

Figure 19-11. Destination Attributes Registers (DATR*n*)

Table 19-11 describes the fields of the DATR*n*.

Table 19-11. DATR*n* Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2	DBPATMU	Bypass ATMU for this DMA operation 0 Route the operation through the ATMU outbound windows. DATR*n*[DWRITETTYPE] should specify a local address space transaction type. 1 Bypass ATMU. Never generate an address match. Always use the attributes in the destination attribute registers. Route the transaction to the interface specified in the DATR*n*[DTRANSINT] field. Applicable only to RapidIO interface.
3	—	Reserved
4–5	DTFLOWLVL	RapidIO transaction flow level 00 Lowest priority transaction flow 01 Next highest priority transaction flow 10 Highest priority transaction flow 11 Reserved Applicable only to RapidIO interface, while DATR*n*[DBPATMU] is set.
6	DPCI_ORDER	PCI ordering rules enable. Applicable only while DATR*n*[DBPATMU] is set. 0 Retain original transaction ordering 1 Follow PCI transaction ordering rules on RapidIO (elevate write priority one level over reads).
7	DSME	Destination stride mode enable 0 Stride mode disabled 1 Stride mode enabled Ignored in basic mode (MR*n*[XFE] is cleared). Striding on the destination address can be accomplished by setting DSME and setting the desired stride size and distance in DSR*n*.
8–11	—	Reserved

Table 19-11. DATR*n* Field Descriptions (continued)

Bits	Name	Description
12–15	DWRITETTYPE	DMA destination transaction type. Reserved values result in a programming error being detected and logged in SR[PE]. Transaction type to run on local address space 0000–0011 Reserved 0100 Write, do not snoop local processor 0101 Write, snoop local processor 0110 Reserved 0111 Reserved 1000–1111 Reserved
16–27	—	Reserved
28–31	EDAD	Extended destination address. EDAD represents the four high-order bits of the 36-bit destination address.

19.3.1.7 Destination Address Registers (DAR*n*)

The destination address registers, shown in Figure 19-12, contain the addresses to which the DMA controller writes data.

In direct mode, if MR*n*[SRW] is set and MR*n*[CDSM/SWSM] is cleared, a write to this register simultaneously sets MR*n*[CS], starting a DMA transfer. Software must ensure that this is a valid address.

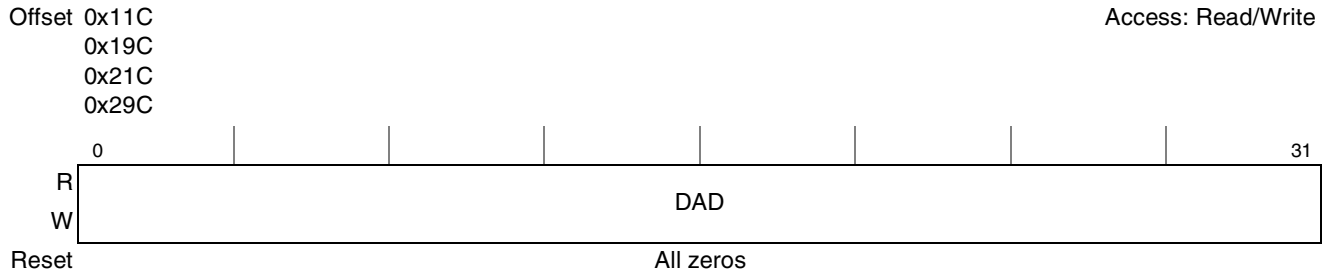


Figure 19-12. Destination Address Registers (DAR*n*)

Table 19-12 describes the field of the DAR*n*.

Table 19-12. DAR*n* Field Descriptions

Bits	Name	Description
0–31	DAD	Destination address. This register contains the destination address of the DMA transfer. The contents are updated after every DMA write operation unless the final stride of a striding operation is less than the stride size, in which case it remains equal to the address from which the last stride began.

19.3.1.8 Byte Count Registers (BCR*n*)

The byte count register, shown in Figure 19-13, contains the number of bytes to transfer.

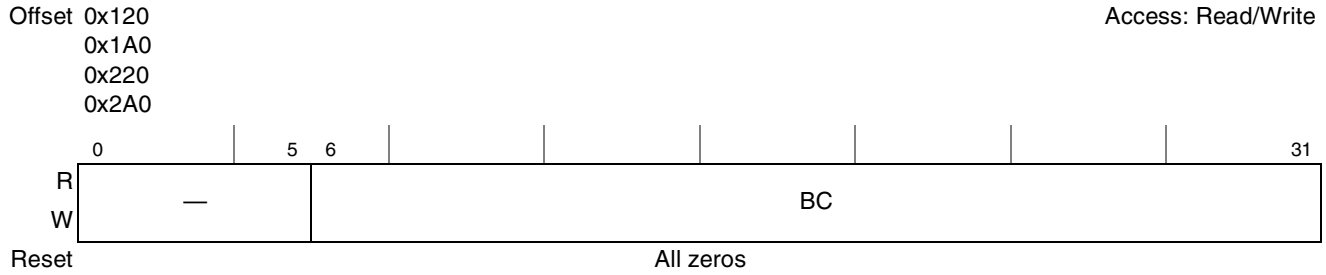


Figure 19-13. Byte Count Registers (BCR*n*)

Table 19-13 describes the fields of the BCR*n*.

Table 19-13. BCR*n* Field Descriptions

Bits	Name	Description
0–5	—	Reserved
6–31	BC	Byte count. Contains the number of bytes to transfer. The value in this register is decremented after each DMA read operation. The maximum transfer size is $(2^{26}) – 1$ bytes.

19.3.1.9 Next Link Descriptor Address Registers (NLNDAR*n* and ENLNDAR*n*)

The next link descriptor address registers, shown in Figure 19-14 and Figure 19-15, contain the address for the next link descriptor in memory. Contents transferred to the current descriptor address registers become effective for the current transfer in basic and extended chaining modes.

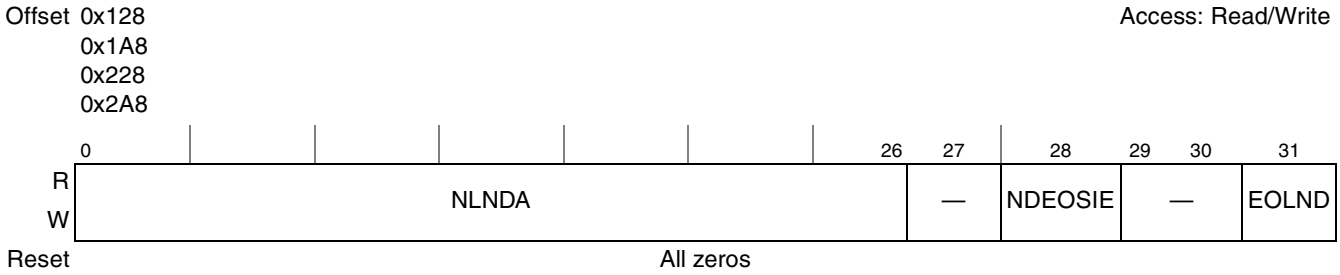


Figure 19-14. Next Link Descriptor Address Registers (NLNDAR*n*)

Table 19-14 describes the fields of the NLNDAR*n* registers.

Table 19-14. NLNDAR*n* Field Descriptions

Bits	Name	Description
0–26	NLNDA	Next link descriptor address. Contains the next link descriptor address in memory. The descriptor must be aligned to a 32-byte boundary.
27	—	Reserved

Table 19-14. NLNDAR*n* Field Descriptions (continued)

Bits	Name	Description
28	NDEOSIE	Next descriptor end-of-segment interrupt enable 0 Do not generate an interrupt if the current DMA transfer for the current descriptor is finished. 1 Generate an interrupt if the current DMA transfer for the current descriptor is finished.
29–30	—	Reserved
31	EOLND	End-of-links descriptor. This bit is ignored in direct mode. 0 This descriptor is not the last link descriptor in memory for this list. 1 This descriptor is the last link descriptor in memory for this list. If this bit is set, the DMA controller advances to the next list descriptor in memory if NLSDAR*n*[EOLSD] is also set in extended mode.

Table 19-14 describes the fields of the NLNDAR*n* registers.

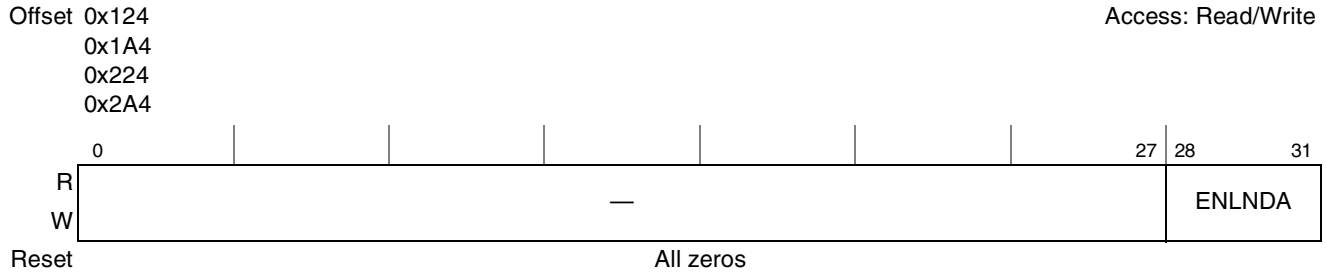


Figure 19-15. Extended Next Link Descriptor Address Registers (ENLNDAR*n*)

Table 19-15 describes the fields of the ENLNDAR*n* registers.

Table 19-15. ENLNDAR*n* Field Descriptions

Bit	Name	Description
0–27	—	Reserved
28–31	ENLNDA	Next link descriptor extended address bits (upper 4 bits of 36-bit address)

19.3.1.10 Current List Descriptor Address Registers (CLSDAR*n* and ECLSDAR*n*)

The current list descriptor address registers, shown in Figure 19-17 and Table 19-17, contain the current address of the list descriptor in memory in extended chaining mode.

In extended chaining mode, software must initialize CLSDAR*n* and ECLSDAR*n* to point to the first list descriptor in memory. After finishing the last link descriptor in the current list, the DMA controller loads the contents of the next list descriptor address register into the current list descriptor address register. If NLSDAR*n*[EOLSD] in the next list descriptor address register is clear, the DMA controller reads the new

current list descriptor from memory to process that list. If EOLSD in the next list descriptor address register is set and the last link in the current list is finished, all DMA transfers are complete.

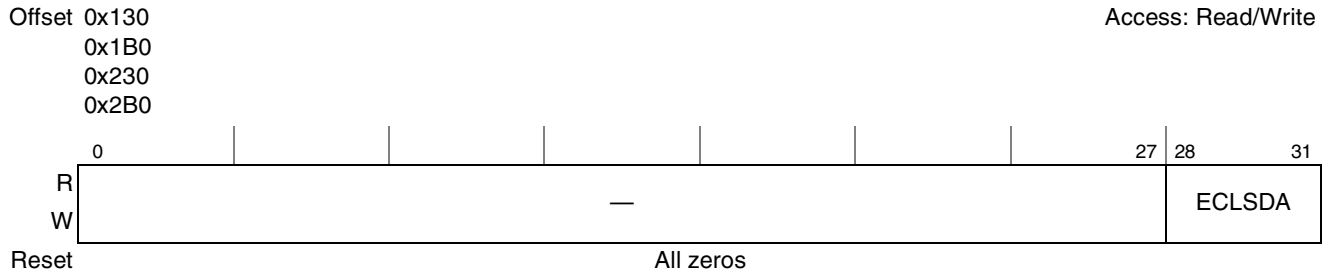


Figure 19-16. Extended Current List Descriptor Address Registers (ECLSDAR*n*)

Table 19-16 describes the fields of the ECLSDAR*n* registers.

Table 19-16. ECLSDAR*n* Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28–31	ECLSDA	Current list descriptor extended address bits (upper 4 bits of 36-bit address)

Figure 19-17 describes the definition for the CLSDAR*n* registers.

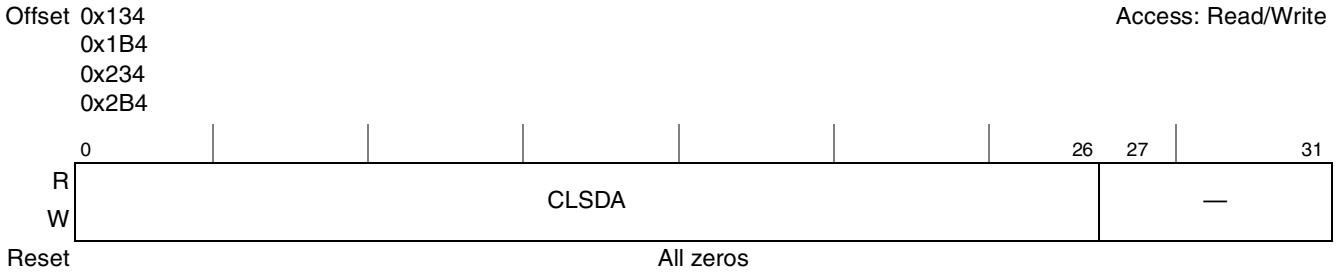


Figure 19-17. Current List Descriptor Address Registers (CLSDAR*n*)

Table 19-17 describes the fields of the CLSDAR*n*.

Table 19-17. CLSDAR*n* Field Descriptions

Bits	Name	Description
0–26	CLSDA	Current list descriptor address. Contains the low-order bits of the 36-bit current list descriptor address of the buffer descriptor in memory in extended chaining mode. The descriptor must be aligned to a 32-byte boundary.
27–31	—	Reserved

19.3.1.11 Next List Descriptor Address Registers (NLSDAR*n* and ENLSDAR*n*)

The next list descriptor address registers, shown in Figure 19-18 and Figure 19-19, contain the address for the next list descriptor in memory. If the contents are transferred to the current list descriptor address register, they become effective for the current transfer in extended chaining mode.

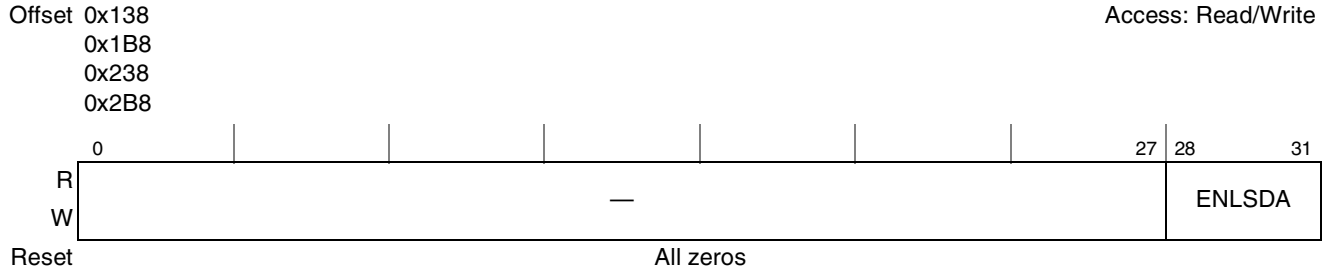


Figure 19-18. Extended Next List Descriptor Address Registers (ENLSDAR*n*)

Table 19-17 describes the fields of the ENLSDAR*n*.

Table 19-18. ENLSDAR*n* Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28–31	ENLSDA	Next list descriptor extended address bits (upper 4 bits of 36-bit address)

Figure 19-19 describes the definition for the NLSDAR*n* registers.

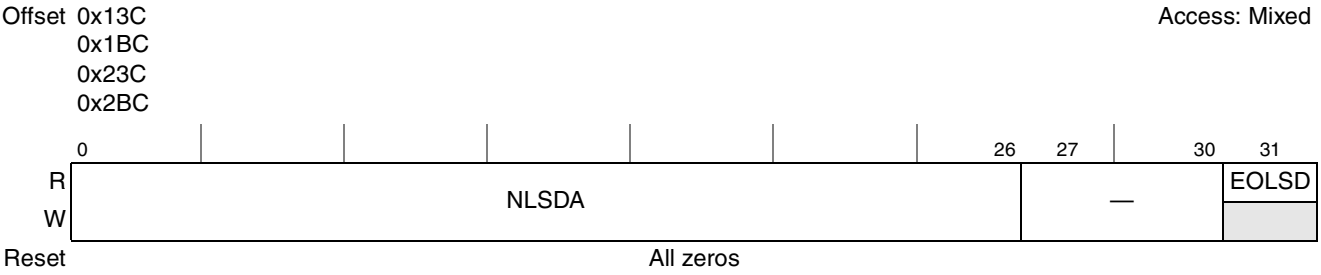


Figure 19-19. Next List Descriptor Address Registers (NLSDAR*n*)

Table 19-19 describes the fields of the NLSDAR*n*.

Table 19-19. NLSDAR*n* Field Descriptions

Bits	Name	Description
0–26	NLSDA	Next list descriptor address. Contains the low-order bits of the 36-bit next descriptor address of the buffer descriptor in memory. The descriptor must be aligned on a 32-byte boundary.
27–30	—	Reserved
31	EOLSD	End-of-lists descriptor. This bit is ignored in direct mode. 0 This list descriptor is not the last list descriptor in memory. 1 This list descriptor is the last list descriptor in memory. If this bit is set, then the DMA controller halts after the last link descriptor transaction is finished.

19.3.1.12 Source Stride Registers (SSR*n*)

The source stride register, shown in Figure 19-20, contains the stride size and distance. Note that the source stride information is loaded when a new list descriptor is read from memory. Therefore, the source stride register is applicable for all link descriptors in the new list. Changing the source stride information for a link requires that a new list be generated.

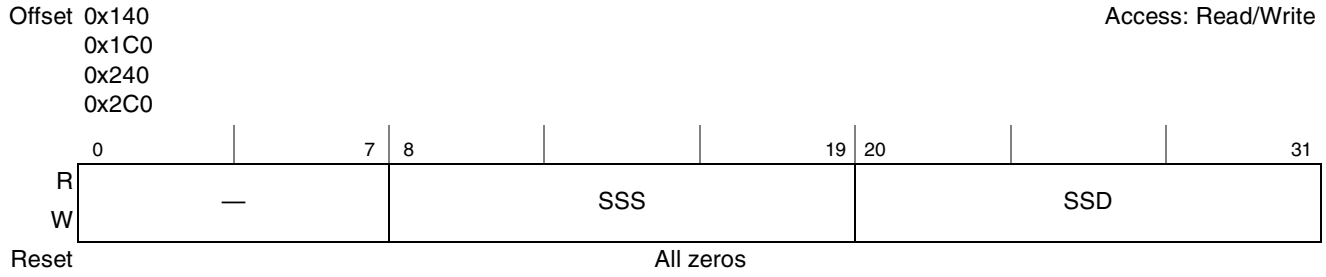


Figure 19-20. Source Stride Registers (SSR*n*)

Table 19-20 describes the fields of the SSR*n*.

Table 19-20. SSR*n* Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–19	SSS	Source stride size. Number of bytes to transfer before jumping to the next address as specified in the source stride distance field.
20–31	SSD	Source stride distance. The source stride distance in bytes from start byte to start byte.

19.3.1.13 Destination Stride Registers (DSR*n*)

The destination stride register contains the stride size, and distance. Note that the destination stride information is loaded when a new list descriptor is read from memory. Therefore, the destination stride register is applicable for all link descriptors in the new list. Changing the destination stride information for a link requires that a new list be generated. Figure 19-21 describes the DSR*n*.

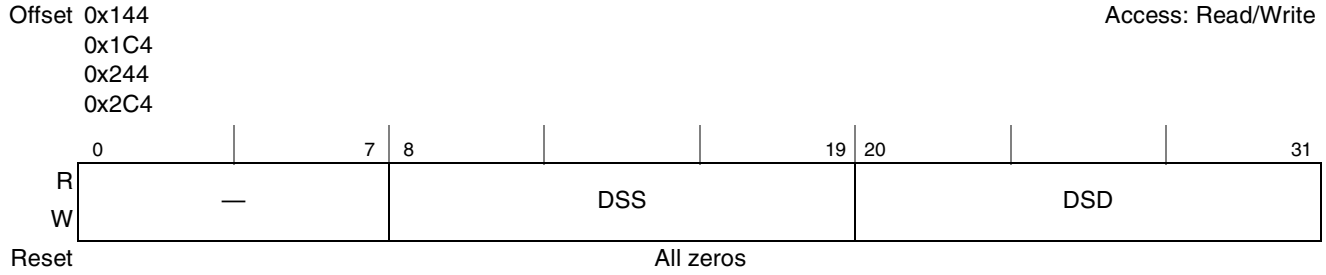


Figure 19-21. Destination Stride Registers (DSR*n*)

Table 19-21 describes the fields of the DSR*n*.

Table 19-21. DSR*n* Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–19	DSS	Destination stride size. Number of bytes to transfer before jumping to the next address as specified in the destination stride distance field.
20–31	DSD	Destination stride distance. The destination stride distance in bytes from start byte to start byte.

19.3.1.14 DMA General Status Register (DGSR)

The DMA general status register combines all of the status bits from each channel into one register. This register is read-only. Figure 19-22 describes the DGSR.

Offset 0x300 Access: Read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TE0	—	CH0	PE0	EOLNI0	CB0	EOSI0	EOLSI0	TE1	—	CH1	PE1	EOLNI1	CB1	EOSI1	EOLSI1
W																
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TE2	—	CH2	PE2	EOLNI2	CB2	EOSI2	EOLSI2	TE3	—	CH3	PE3	EOLNI3	CB3	EOSI3	EOLSI3
W																
Reset	All zeros															

Figure 19-22. DMA General Status Register (DGSR)

Table 19-22 describes the fields of the DGSR.

Table 19-22. DGSR Field Descriptions

Bits	Name	Description
0	TE0	Transfer error, channel 0 0 Normal operation 1 An error condition occurred during the DMA transfer.
1	—	Reserved
2	CH0	Channel halted, channel 0
3	PE0	Programming error, channel 0
4	EOLNI0	End-of-links interrupt, channel 0
5	CB0	Channel busy, channel 0
6	EOSI0	End-of-segment interrupt, channel 0
7	EOLSI0	End-of-lists/direct interrupt, channel 0
8	TE1	Transfer error, channel 1 0 Normal operation 1 An error condition occurred during the DMA transfer.
9	—	Reserved

Table 19-22. DGSR Field Descriptions (continued)

Bits	Name	Description
10	CH1	Channel halted, channel 1
11	PE1	Programming error, channel 1
12	EOLNI1	End-of-links interrupt, channel 1
13	CB1	Channel busy, channel 1
14	EOSI1	End-of-segment interrupt, channel 1
15	EOLSI1	End-of-lists/direct interrupt, channel 1
16	TE2	Transfer error, channel 2 0 Normal operation 1 An error condition occurred during the DMA transfer.
17	—	Reserved
18	CH2	Channel halted, channel 2
19	PE2	Programming error, channel 2
20	EOLNI2	End-of-links interrupt, channel 2
21	CB2	Channel busy, channel 2
22	EOSI2	End-of-segment interrupt, channel 2
23	EOLSI2	End-of-lists/direct interrupt, channel 2
24	TE3	Transfer error, channel 3 0 Normal operation 1 An error condition occurred during the DMA transfer.
25	—	Reserved
26	CH3	Channel halted, channel 3
27	PE3	Programming error, channel 3
28	EOLNI3	End-of-links interrupt, channel 3
29	CB3	Channel busy, channel 3
30	EOSI3	End-of-segment interrupt, channel 3
31	EOLSI3	End-of-lists/direct interrupt, channel 3

19.4 Functional Description

This section describes the function of the DMA controller.

19.4.1 DMA Channel Operation

All DMA channels support two different modes of operation: a basic mode (MR*n*[XFE] is cleared) and an extended mode (MR*n*[XFE] is set). In both modes, a channel can be activated by clearing and setting MR*n*[CS], or through the single-write start mode using MR*n*[CDSM/SWSM] and MR*n*[SRW], or through an external control mode using MR*n*[ECS_EN].

In basic mode, the channel can be programmed in basic direct mode or basic chaining mode. In extended mode, the channel can be programmed in extended direct mode or extended chaining mode. Extended mode provides more capabilities, such as extended descriptor chaining, striding capabilities, and a more flexible descriptor structure.

The DMA controller supports misaligned transfers for both the source and destination addresses. In order to maximize performance, the source and destination engines align the source and destination addresses to a 64-byte boundary. The DMA always reads/writes the maximum number of bytes for a given transfer as described by the capability inputs of the DMA controller except for globally coherent transactions that use the size of the cache coherence granule as described by the mode select input.

The DMA controller supports bandwidth control, which prevents a channel from consuming all the data bandwidth in the controller. Each channel is allowed to consume the bandwidth of the shared resources as specified by the bandwidth control value. After the channel uses its allotted bandwidth, the arbiter grants the next channel access to the shared resources. The arbitration is round robin between the channels. This feature is also used to implement the external control pause feature. If the external control start and pause are enabled in the MR*n*, the channel enters a paused state after transferring the data described in the bandwidth control. External control can restart the channel from a paused state.

The DMA programming model permits software to program each DMA engine independently to interrupt on completed segment, chain, or error. It also provides the capability for software to resume the DMA engine from a hardware halted condition by setting the channel continue bit, MR*n*[CC]. See Table 19-23 for more complete descriptions of the channel states and state transitions.

19.4.1.1 Basic DMA Mode Transfer

This mode is primarily included for backward compatibility with existing DMA controllers which use a simple programming model. This is the default mode out of reset. The different modes of operation under the basic mode are explained in the following sections.

19.4.1.1.1 Basic Direct Mode

In basic direct mode, the DMA controller does not read descriptors from memory, but instead uses the current parameters programmed in the DMA registers to start the DMA transfer. Software is responsible for initializing SAR*n*, SATR*n*, DAR*n*, DATR*n*, and BCR*n* registers. The DMA transfer is started when MR*n*[CS] is set. Software is expected to program all the appropriate registers before setting MR*n*[CS] to a 1. The transfer is finished after all the bytes specified in the byte count register have been transferred or if an error condition occurs. The sequence of events to start and complete a transfer in basic direct mode is as follows:

1. Poll the channel state (see Table 19-23), to confirm that the specific DMA channel is idle.
2. Initialize SAR*n*, SATR*n*, DAR*n*, DATR*n* and BCR*n*.
3. Set the mode register channel transfer mode bit, MR*n*[CTM], to indicate direct mode. Other control parameters may also be initialized in the mode register.
4. Clear, then set the mode register channel start bit, MR*n*[CS], to start the DMA transfer.
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.

6. SR*n*[CB] is automatically cleared by the DMA controller after the transfer is finished, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if a transfer error occurs.
7. End of segment interrupt is generated if MR*n*[EOSIE] is set.

19.4.1.1.2 Basic Direct Single-Write Start Mode

In basic direct single-write start mode, the DMA controller does not read descriptors from memory, but instead uses the current parameters programmed in the DMA registers to start the DMA transfer. Software is responsible for initializing the SATR*n*, DATR*n*, and BCR*n* registers. Setting MR*n*[SRW] configures the DMA controller to begin the DMA transfer either when SAR*n* is written or when DAR*n* is written, determined by the state of MR*n*[CDSM/SWSM]. Writing to SAR*n* initiates the DMA transfer if MR*n*[CDSM/SWSM] is set. Writing to DAR*n* initiates the DMA transfer if MR*n*[CDSM/SWSM] is cleared. The DMA controller automatically sets the channel start bit, MR*n*[CS]. Software is expected to program all the appropriate registers before writing the source or destination address registers. The transfer is finished after all the bytes specified in the byte count register have been transferred or if an error condition occurs. The sequence of events to start and complete a transfer in single-write start basic direct mode is as follows:

1. Poll the channel state (see Table 19-23) to confirm that the specific DMA channel is idle.
2. Initialize the source attributes (SATR*n*), DATR*n*, and BCR*n* registers.
3. Set the mode register channel transfer mode bit, MR*n*[CTM], and the single-write start direct mode bit, MR*n*[SRW]. Other control parameters may also be initialized in the mode register. Set MR*n*[CDSM/SWSM] for transfers started using SAR*n*. Clear MR*n*[CDSM/SWSM] for transfers started using the DAR*n*.
4. A write to the source or destination address register starts the DMA transfer and automatically sets MR*n*[CS].
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
6. SR*n*[CB] is automatically cleared by the DMA controller after the transfer is finished, if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if a transfer error occurs.
7. End of segment interrupt is generated if MR*n*[EOSIE] is set.

19.4.1.1.3 Basic Chaining Mode

In basic chaining mode, software must first build link descriptor segments in memory. Then the current link descriptor address register must be initialized to point to the first descriptor in memory. The DMA controller loads descriptors from memory prior to a DMA transfer. The DMA controller begins the transfer according to the link descriptor information loaded for the segment. After the current segment is finished, the DMA controller reads the next link descriptor from memory and begins another DMA transfer. The transfer is finished if the current link descriptor is the last one in memory or if an error condition occurs. The sequence of events to start and complete a transfer in chaining mode is as follows:

1. Build link descriptor segments in memory.
2. Poll the channel state (see Table 19-23) to confirm that the specific DMA channel is idle.
3. Initialize CLNDAR*n* and ECLNDAR*n* to point to the first link descriptor in memory.

4. Clear the mode register channel transfer mode bit, MR*n*[CTM], as well as MR*n*[XFE], to indicate basic chaining mode. Other control parameters may also be initialized in the mode register.
5. Clear and then set the mode register channel start bit, MR*n*[CS], to start the DMA transfer.
6. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
7. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

19.4.1.1.4 Basic Chaining Single-Write Start Mode

Basic chaining single-write start mode allows a chain to be started by writing the current link descriptor address register (CLNDAR*n*). (Note that ECLNDAR*n* must be written *first* so that the full 36-bit descriptor address is present when the chain starts.) Setting MR*n*[CDSM/SWSM] in the mode register causes MR*n*[CS] to be automatically set when the current link descriptor address register is written. The sequence of events to start and complete a chain using single-write start mode is as follows:

1. Set the mode register current descriptor start mode bit, MR*n*[CDSM/SWSM], and the extended features enable bit MR*n*[XFE]. Also, clear the channel transfer mode bit, MR*n*[CTM]. This initialization indicates basic chaining and single-write start mode. Also other control parameters may be initialized in the mode register.
2. Build link descriptor segments in memory.
3. Poll the channel state (see Table 19-23) to confirm that the specific DMA channel is idle.
4. Initialize CLNDAR*n* and ECLNDAR*n* to point to the first descriptor segment in memory. This write automatically causes the DMA controller to begin the link descriptor fetch and set MR*n*[CS].
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
6. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

19.4.1.2 Extended DMA Mode Transfer

The extended DMA mode also operates in chaining and direct mode. It offers additional capability over the basic mode by supporting striding and a more flexible descriptor structure. This additional functionality also requires a new and more complex programming model. The extended DMA mode is activated by setting MR*n*[XFE].

19.4.1.2.1 Extended Direct Mode

Extended direct mode has the same functionality as basic direct mode with the addition of stride capabilities. The bit settings are the same as in direct mode with the exception of the MR*n*[XFE] being set. Striding on the source address can be accomplished by setting SATR*n*[SSME] and setting the desired stride size and distance in SSR*n*. Striding on the destination address can be accomplished by setting DATR*n*[DSME] and setting the desired stride size and distance in DSR*n*.

19.4.1.2.2 Extended Direct Single-Write Start Mode

Extended direct single-write start mode has the same functionality as the basic direct single-write start mode with the addition of stride capabilities. The bit settings are also the same with the exception of MR*n*[XFE] being set. Striding on the source address can be accomplished by setting SATR*n*[SSME] and setting the desired stride size and distance in SSR*n*. Striding on the destination address can be accomplished by setting DATR*n*[DSME] and setting the desired stride size and distance in DSR*n*.

19.4.1.2.3 Extended Chaining Mode

In extended chaining mode, the software must first build list and link descriptor segments in memory. Then CLSDAR*n* and ECLSDAR*n* must be initialized to point to the first list descriptor in memory. The DMA controller loads list descriptors and link descriptors from memory prior to a DMA transfer. The DMA controller begins the transfer according to the link descriptor information loaded. Once the current link descriptor is finished, the DMA controller reads the next link descriptor from memory and begins another DMA transfer. If the current link descriptor is the last in the list, the DMA controller reads the next list descriptor in memory. The transfer is finished if the current link descriptor is the last one in the last list in memory or if an error condition occurs. The sequence of events to start and complete a transfer in extended chaining mode is as follows:

1. Build link and list descriptor segments in memory.
2. Poll the channel state (see Table 19-23) to confirm that the specific DMA channel is idle.
3. Initialize CLSDAR*n* and ECLSDAR*n* to point to the first list descriptor in memory.
4. Clear the mode register channel transfer mode bit, MR*n*[CTM], to indicate chaining mode. MR*n*[XFE] must be set to indicate extended DMA mode. Other control parameters may also be initialized in the mode register.
5. Clear and then set the mode register channel start bit, MR*n*[CS], to start the DMA transfer.
6. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
7. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

19.4.1.2.4 Extended Chaining Single-Write Start Mode

In the extended mode, the single-write start feature allows a chain to be started by writing the current list descriptor pointer. Setting MR*n*[CDSM/SWSM] causes MR*n*[CS] to be set automatically when CLSDAR*n* is written. (Note that ECLSDAR*n* must be written *first* so that the full 36-bit descriptor address is present when the chain starts.) The sequence of events to start and complete an extended chain using single-write start mode is as follows:

1. Set MR*n*[CDSM/SWSM] and MR*n*[XFE] and clear MR*n*[CTM] to indicate extended chaining and single-write start mode. Also other control parameters may be initialized in the mode register.
2. Build list and link descriptor segments in local memory.
3. Poll the channel state (see Table 19-23) to confirm that the specific DMA channel is idle.

4. Initialize the current list descriptor address register to point to the first list descriptor segment in memory. This write automatically causes the DMA controller to begin the list descriptor fetch and set MR*n*[CS].
5. SR*n*[CB] is set by the DMA controller to indicate the DMA transfer is in progress.
6. SR*n*[CB] is automatically cleared by the DMA controller after finishing the transfer of the last descriptor segment, or if the transfer is aborted (MR*n*[CA] transitions from a 0 to 1), or if an error occurs during any of the transfers.

19.4.1.3 External Control Mode Transfer

An external control can be used to control all DMA channels by setting MR*n*[EMS_EN]. The external control can direct the DMA channel in the following transfer modes:

- Basic direct
- Basic chaining
- Extended direct
- Extended chaining

Note that when operating the DMA in chaining mode the register byte count field, BCR[BC], must be initialized to zero before enabling the pause feature. In chaining modes, the channel does not pause for descriptor fetch transfer.

The external control and the DMA controller use a well defined protocol to communicate. The external control can start or restart a paused DMA transfer. The DMA controller acknowledges a DMA transfer in progress and also indicates a transfer completion. Note that external control cannot cause a channel to enter a paused state.

The pause feature can be enabled by setting MR*n*[EMP_EN]. MR*n*[BWC] specifies how much data to allow a specific channel to transfer before entering a paused state by clearing MR*n*[CS]. Note however, that write data for a paused transfer may not have reached the target interface when so indicated. The channel can be restarted from a paused state by the asserted edge of $\overline{\text{DREQ}}$ as driven by an external master. In chaining modes, the channel does not pause for descriptor fetch transfer; it only pauses during the actual data transfer.

The following signals are defined for the external control interface:

- $\overline{\text{DMA\_DREQ}}$—Asserting edge triggers a DMA transfer start or restart from a pause request. Sets MR*n*[CS]. (Note that negating $\overline{\text{DMA\_DREQ}}$ does NOT clear MR*n*[CS].)
- $\overline{\text{DMA\_DACK}}$—Indicates a DMA transfer currently in progress. SR*n*[CB] is set.
- $\overline{\text{DMA\_DDONE}}$—Indicates the completion of the DMA controller's involvement in the transfer and the readiness to accept a new DMA command. SR*n*[CB] is clear. Note however, that write data may still be queued at the target interface or in the process of transfer on an external interface.

Detailed descriptions of the external control interface are in Table 19-3. The timing diagram of the external control interface is shown in Figure 19-23.

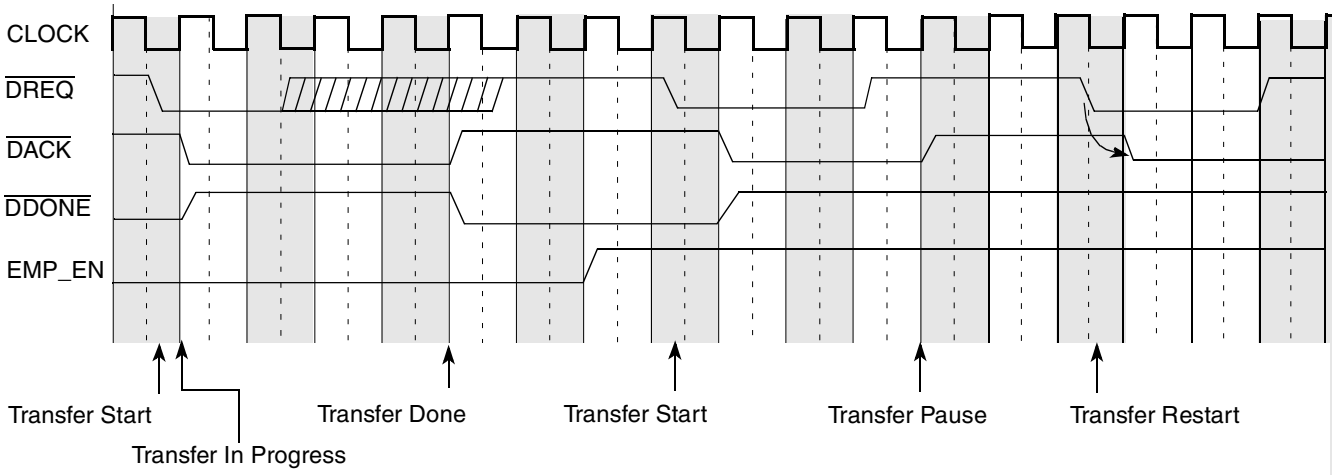


Figure 19-23. External Control Interface Timing

19.4.1.4 Channel Continue Mode for Cascading Transfer Chains

The channel continue mode (enabled when MR*n*[CC] is set) offers software the flexibility of having the DMA controller start on descriptors that have already been programmed while software continues to build more descriptors in memory. Software can set the end-of-links descriptor (EOLND) in basic mode, or end-of-lists descriptor (EOLSD) in extended mode, to cause the channel to go into a halted state while software continues to build other descriptors in memory. Software can then set CC to force hardware to continue where it left off. Channel continue is only meaningful for chaining modes, not direct mode.

If CC is set by software while the channel is busy with a transfer, the DMA controller finishes all transfers until it reaches the EOLND in basic mode or EOLSD in extended mode. The DMA controller then refetches the last link descriptor in basic mode or the last list descriptor in extended mode and clears the channel continue bit. If EOLND or EOLSD is still set for their respective modes, the DMA controller remains in the idle state. If EOLND or EOLSD is not set, the DMA controller continues the transfer by refetching the new descriptor. The channel busy (SR*n*[CB]) bit is cleared when the DMA controller reaches EOLND/EOLSD and is set again when it initiates the refetch of the link or list descriptor.

If CC is set by software while the channel is not busy with a transfer, the DMA controller refetches the last link descriptor in basic mode, or the last list descriptor in extended mode and clears the channel continue bit. If EOLND or EOLSD is still set for their respective modes, the DMA controller remains in the idle state. If the EOLND or EOLSD bits are not set, the DMA controller continues the transfer by refetching the new descriptor.

19.4.1.4.1 Basic Mode

On a channel continue, the descriptor at the current link descriptor address registers (CLNDAR*n* and ECLNDAR*n*) is refetched to get the next link descriptor address field as updated by software. The channel halts if NLNDAR*n*[EOLND] is still set. If EOLND is zero, the next link descriptor address is copied into CLNDAR*n* and ECLNDAR*n* and the channel continues with another descriptor fetch of the current link

descriptor address. As a result, two link descriptor fetches always exist after channel continue before starting the first transfer.

19.4.1.4.2 Extended Mode

On a channel continue, the descriptor at the current list descriptor (CLSDAR*n* and ECLSDAR*n*) address register is refetched to get the next list descriptor address field as updated by software. The channel halts if NLSDAR*n*[EOLSD] is still set. If not, the next list descriptor address is copied into the CLSDAR*n* and ECLSDAR*n* registers and the channel continues with another descriptor fetch of the current list descriptor address. As a result, two list descriptor fetches always exist after channel continue before the first link descriptor fetch and the first transfer.

19.4.1.5 Channel Abort

Software can abort a previously initiated transfer by setting MR*n*[CA]. Once the DMA channel controller detects a zero-to-one transition of MR*n*[CA], it finishes the current sub-block transfer and halts all further activity. The controller then waits for all previously initiated transfers from the specified channel to drain and clears SR*n*[CB]. Successful completion of a software initiated abort request can be recognized by MR*n*[CA] being set and SR*n*[CB] being cleared. Obviously, if the controller was already halted because of an error condition (SR*n*[TE] is set), or the channel has completed all transfers, then SR*n*[CB] being cleared may not signify that the controller entered a halt state due to the abort request.

19.4.1.6 Bandwidth Control

MR*n*[BWC] specifies how much data to allow a specific channel to transfer before allowing the next channel to use the shared data transfer hardware. This promotes equitable bandwidth allocation between channels. However, if only one channel is busy, hardware overrides the specified bandwidth control size value. The DMA controller allows a channel to transfer up to 1 Kbyte at a time when no other channel is active.

19.4.1.7 Channel State

Table 19-23 defines the state of a channel based on the values of the channel start (MR*n*[CS]), channel busy (SR*n*[CB]), transfer error (SR*n*[TE]), and channel continue (MR*n*[CC]) bits.

Table 19-23. Channel State Table

MR*n*[CS]	SR*n*[CB]	SR*n*[TE]	MR*n*[CC]	Channel State
0	0	0	0	Idle state. This is the state of the bits out of reset.
0	0	0	1	Channel continue unexpected. Channel remains idle
0	0	1	0	Error occurred after software halted the channel.
0	0	1	1	Channel continue unexpected. Channel remains in error halt state
0	1	0	0	Software halted channel. The channel was busy and software cleared MR*n*[CS].
0	1	0	1	Channel remains in halt state.
—	1	1	—	The channel has encountered an error condition and it is trying to halt.

Table 19-23. Channel State Table (continued)

MR*n*[CS]	SR*n*[CB]	SR*n*[TE]	MR*n*[CC]	Channel State
1	0	0	0	Ready to start a transfer, or transfer completed
1	0	0	1	Continue transfer (only meaningful in chaining mode, not direct mode). In direct mode, the channel continue has no effect.
1	0	1	0	Error occurred during transfer
1	0	1	1	Channel remains in error halt state
1	1	0	0	Transfer in progress
1	1	0	1	Continue after reaching the end of list/link, or the first descriptor fetch after channel continue

19.4.1.8 Illustration of Stride Size and Stride Distance

If operating in stride mode, the stride size defines the amount of data to transfer before jumping to the next quantity of data as specified by the stride distance. The stride distance is added to the current base address to point to the next quantity of data to be transferred. Figure 19-24 illustrates the stride size and distance parameters. As shown, each time the stride distance is added to the base address, the resulting address becomes the new base address. This sequence repeats until the amount of data transferred equals the transfer size.

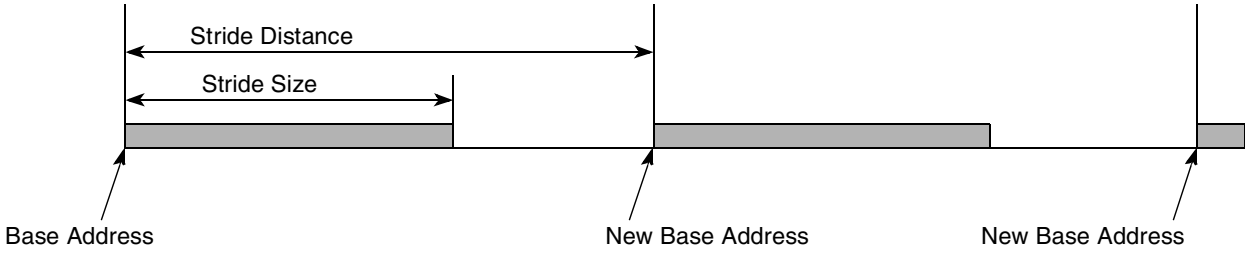


Figure 19-24. Stride Size and Stride Distance

19.4.2 DMA Transfer Interfaces

The DMA can be used to achieve data transfers across the entire memory map. Note that a single DMA transfer in any of the direct or chaining modes must not cross a 16GB (34-bit) address boundary.

19.4.3 DMA Errors

On a transfer error (non-correctable ECC errors on memory accesses, parity errors on local bus or PCI, address mapping errors, for example), the DMA halts by setting SR*n*[TE] and generates an interrupt if MR*n*[EIE] is set. On a programming error, the DMA sets SR*n*[PE] and generates an interrupt if MR*n*[EIE] is set. The DMA controller detects the following programming errors:

- Transfer started with a byte count of zero
- Stride transfer started with a stride size of zero
- Transfer started with a priority of three

- Illegal type, defined by SATR*n*[SREADTTYPE] and DATR*n*[DWRITETTYPE], used for the transfer.
- Invalid interface—Defined by SATR*n*[STRANSINT] and DATR*n*[DTRANSINT]. Used for the transfer when in ATMU bypass mode.

19.4.4 DMA Descriptors

The DMA engine recognizes list descriptors and link descriptors. List descriptors connect lists of link descriptors. Link descriptors describe the DMA activity that is to take place. DMA descriptors are built in either local or remote memory and are connected by the next descriptor fields. Only link descriptors contain information for the DMA controller to transfer data. Software must ensure that each descriptor is 32-byte aligned. The last link descriptor in the last list in memory sets the NLNDAR*n*[EOLND] bit in the next link descriptor and NLSDAR*n*[EOLSD] in the next list descriptor fields indicating that these are the last descriptors in memory. Software initializes the current list descriptor address register to point to the first list descriptor in memory. The DMA controller traverses through the descriptor lists until the last link descriptor is met. For each link descriptor in the chain, the DMA controller starts a new DMA transfer with the control parameters specified by that descriptor. Link and list descriptor fetches always snoop the local memory space.

NOTE

Software must ensure that each descriptor is aligned on a 32-byte boundary.

Table 19-24 summarizes the DMA list descriptors.

Table 19-24. List DMA Descriptor Summary

Descriptor Field	Description
Next list descriptor extended address	Points to the next list descriptor in memory. After the DMA controller reads the descriptor from memory, this field is loaded into the next list descriptor extended address registers.
Next list descriptor address	Points to the next list descriptor in memory. After the DMA controller reads the descriptor from memory, this field is loaded into the next list descriptor address registers.
First link descriptor extended address	Points to the first link descriptor in memory for this list. After the DMA controller reads the descriptor from memory, this field is loaded into the current link descriptor extended address registers.
First link descriptor address	Points to the first link descriptor in memory for this list. After the DMA controller reads the descriptor from memory, this field is loaded into the current link descriptor address registers.
Source stride	Contains the stride information used for the data source if striding is enabled for a link in the list
Destination stride	Contains the stride information used for the data destination if striding is enabled for a link in the list

Table 19-25 summarizes the DMA link descriptors.

Table 19-25. Link DMA Descriptor Summary

Descriptor Field	Description
Source attributes register	Contains source transaction attributes
Source address	Contains the source address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the Source address register.

Table 19-25. Link DMA Descriptor Summary (continued)

Descriptor Field	Description
Destination attributes register	Contains destination transaction attributes
Destination address	Contains the destination address of the DMA transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the destination address register.
Next link descriptor extended address	Points to the next link descriptor in memory. After the DMA controller reads the link descriptor from memory, this field is loaded into the extended next link descriptor address registers
Next link descriptor address	Points to the next link descriptor in memory. After the DMA controller reads the link descriptor from memory, this field is loaded into the next link descriptor address registers.
Byte count	Contains the number of bytes to transfer. After the DMA controller reads the descriptor from memory, this field is loaded into the byte count register.

Figure 19-25 describes the DMA transaction flow.

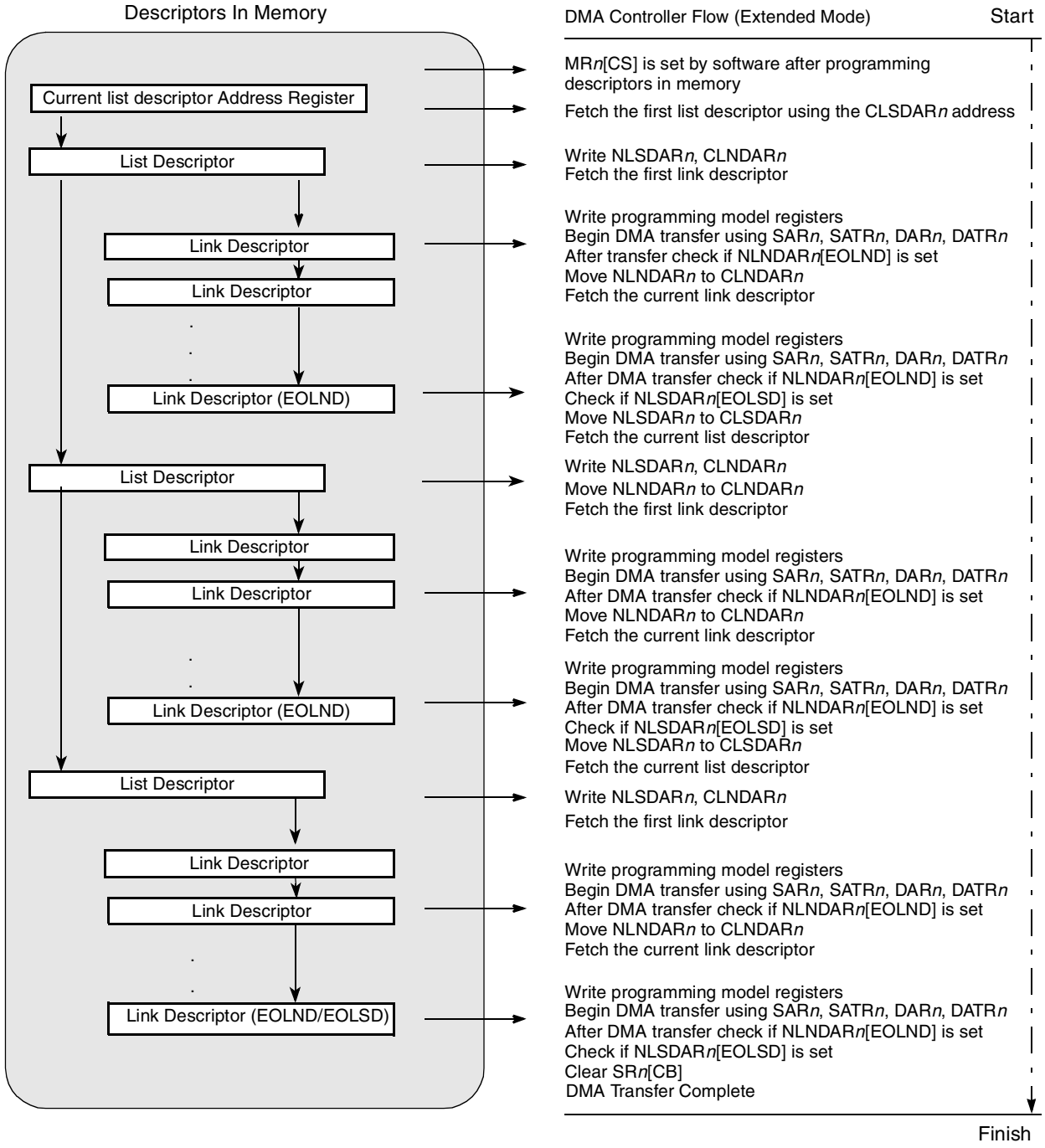


Figure 19-25. DMA Transaction Flow with DMA Descriptors

Figure 19-26 describes the format of the list descriptors.

Offset	
0x00	Next List Descriptor Extended Address
0x04	Next List Descriptor Address
0x08	First Link Descriptor Extended Address
0x0C	First Link Descriptor Address
0x10	Source Stride
0x14	Destination Stride
0x18	Reserved
0x1C	Reserved

Figure 19-26. List Descriptor Format

Figure 19-27 describes the format of the link descriptors.

Offset	
0x00	Source Attributes
0x04	Source Address
0x08	Destination Attributes
0x0C	Destination Address
0x10	Next Link Descriptor Extended Address
0x14	Next Link Descriptor Address
0x18	Byte Count
0x1C	Reserved

Figure 19-27. Link Descriptor Format

19.4.5 Limitations and Restrictions

This section addresses some of the limitations and restrictions of the DMA controller and is intended to help software maximize the DMA performance and avoid DMA programming errors.

The limitations of the DMA controller are the following:

- Due to the limited number of buffers that the DMA controller can use, stride sizes less than 64 bytes should be avoided. Maximum utilization is obtained from strides greater than or equal to 256 bytes. However, small stride sizes can be used for scatter-gather functions.
- Coherent reads or writes are broken up into cache line accesses in the DMA.

The DMA controller restrictions are as follows:

- Setting the source or destination priority level (STFLOWLVL or DTFLOWLVL) to a value of three (0b11) is considered a programming error.
- All interface capabilities from where descriptors are being fetched must support read sizes of 32 bytes or greater.

- If MR*n*[SAHE] is set, the source interface transfer size capability must be greater than or equal to MR*n*[SAHTS]. The source address must be aligned to a size specified by SAHTS.
- If MR*n*[DAHE] is set, the destination interface transfer size capability must be greater than or equal to MR*n*[DAHTS]. The destination address must be aligned to the size specified by DAHTS.
- Destination striding is not supported if MR*n*[DAHE] is set and source striding is not supported if MR*n*[SAHE] is set.
- A single DMA transfer in any of the direct or chaining modes must not cross a 16GB (34-bit) address boundary
- Striding does not work if the destination transaction type is MESSAGE (DATR[DWRITETTYPE] = 0x0110) because messages have no memory addresses. As well, destination address hold should be disabled (MR*n*[DAHE] is cleared) unless the destination address hold transfer size indicates an 8-byte message (MR*n*[DAHTS] = 0x11). Software is responsible for disabling striding and DAHE, in this case, and for ensuring that the bandwidth control is large enough to support the desired message size. Failure to adhere to these restrictions results in boundedly undefined behavior.
- When DMA is used to issue maintenance reads and writes in bypass mode, the sum of the starting offset field in DAR and the byte count must not cause the offset to rollover. Failure to adhere to this restriction results in boundedly undefined behavior.

19.5 DMA System Considerations

This section provides information about how to make most effective use of the DMA channels.

Table 19-26 summarizes the device DMA capabilities.

Table 19-26. DMA Capabilities

DMA Channel	Initiator			
	Software	On-Chip Peripheral[1,2]	On-Chip Peripheral[3]	External Signals[4]
DMA1 — Located on OCN1				
0	Yes	SSI1_TX	IR2_RX	$\overline{\text{DMA1\_DREQ0}}$, $\overline{\text{DMA1\_ACK0}}$, $\overline{\text{DMA1\_DDONE0}}$ (muxed on IRQ[6:8])
1	Yes	SSI1_RX	IR2_TX	—
2	Yes	SSI2_TX	IR1_RX	—
3	Yes	SSI2_RX	IR1_TX	$\overline{\text{DMA1\_DREQ3}}$, $\overline{\text{DMA1\_ACK3}}$, $\overline{\text{DMA1\_DDONE3}}$ (muxed on IRQ[9:11])
DMA2 — Located on OCN2				
0	Yes	SSI1_TX	IR2_RX	$\overline{\text{DMA2\_DREQ0}}$, $\overline{\text{DMA2\_ACK0}}$, $\overline{\text{DMA2\_DDONE0}}$ (Muxed on $\overline{\text{LCS}}$[5:7]))
1	Yes	SSI1_RX	IR2_TX	—

Table 19-26. DMA Capabilities (continued)

DMA Channel	Initiator			
	Software	On-Chip Peripheral[1,2]	On-Chip Peripheral[3]	External Signals[4]
2	Yes	SSI2_TX	IR1_RX	—
3	Yes	SSI2_RX	IR1_TX	$\overline{\text{DMA2\_DREQ3}}$, $\overline{\text{DMA2\_ACK3}}$, $\overline{\text{DMA2\_DDONE3}}$ (Muxed ON GPIO[29:31])

1 For DMA Channel i, selected by setting DMACR[DMA1_i] (DMA1) or DMACR[DMA2_i] (DMA2) to 00

2 SSI DMA requests are generated from TXFIFO0 and RXFIFO0 which supports the SSI's single- and dual-channel modes. No requests are generated from TXFIFO1 and RXFIFO1.

3 For DMA Channel i, selected by setting DMACR[DMA1_i] (DMA1) or DMACR[DMA2_i] (DMA2) to 01

4 For DMA Channel 0, selected by setting PMUXCR[DMA*n*_0] to 1; for DMA Channel 3, selected by setting PMUXCR[DMA*n*_3] to 1, where *n* is the appropriate DMA controller

Table 19-27 shows the matrix of possible DMA transfers, indicating which of these are supported on the device.

Table 19-27. MPC8610 DMA Paths

	DDR	LBC	PCI	PEX1 (×4)	PEX2 (×8)	FIRI/SIRI	SSI
DDR	Y	Y	Y	Y	Y	Y	Y
LBC	Y	Y	Y	Y	Y	Y	Y
PCI	Y	Y	Y	Y	—	Y	Y
PEX1 (×4)	Y	Y	Y	Y	—	Y	Y
PEX2 (×8)	Y	Y	—	—	Y	Y	Y
FIRI/SIRI	Y	Y	Y	Y	Y	Y	Y
SSI	Y	Y	Y	Y	Y	Y	Y

Note: On-chip target configuration registers include I^2C data register.

Note: .On-chip 4-channel controllers can serve external masters.

19.5.1 Unusual DMA Scenarios

The following is a description of unusual DMA paths including explanations of why some functional blocks cannot serve as DMA targets. The following topics are addressed:

- DMA transaction initiators (masters)
- DMA targets, that is, data sources or destinations
- Transparency of the bus controllers to DMA transactions
- What is useful as opposed to what is possible. For example, any register can be addressed through an internal control bus, which means configuration and control registers can be DMA targets.

19.5.1.1 DMA to e600 Core

The L1 cache cannot be a direct DMA target because it cannot be directly addressed by software. However, DMA access into the L1 cache occurs indirectly if a block of memory that is cached in the L1 is specified as the DMA target. This effect is deterministic if the target memory block was locked into the L1 with cache locking instructions.

19.5.1.2 DMA to Configuration, Control, and Status Registers

Because any internal register can be addressed with the four-channel DMA controller, configuration, control, and status registers throughout the device are valid DMA targets. However, the primary purpose of DMA—to reduce processor load by moving large blocks of data—is not served by DMA transfers of configuration data. For example, while it is possible to DMA into the I^2C controller or programmable interrupt controller (PIC), doing so is extremely inefficient and is seldom beneficial in normal operation. The overhead of creating DMA descriptors far exceeds any savings in CPU cycles.

19.5.1.3 DMA to I^2C

The I^2C controller is not transparent to DMA transfers. Observe the caveats listed in Section 19.5.1.2, “DMA to Configuration, Control, and Status Registers,” when accessing any I^2C register, including the data register (I2CDR).

19.5.1.4 DMA to DUART

The DUART provides complete and sophisticated DMA support, which is described in Section 13.4.5, “FIFO Mode.”

19.5.1.5 DMA to SSI

Select DMA channels can be configured, using DMACR and PMUXCR, to respond to SSI requests directly. See Section 23.4.1.22, “DMA Control Register (DMACR),” and Section 23.4.1.9, “Alternate Function Signal Multiplex Control Register (PMUXCR).” The DMA controller only transfers data to SSI based on FIFO availability.

Software sets the FIFO watermarks in the SSI FIFO control register (SFCSR[RFWM and TFWM]) and enables the DMA requests in the SSI interrupt enable register (SIER[RDMAE and TDMAE]. The DMA controller responds to these SSI DMA requests and transmits the predetermined amount of data needed by the SSI. Using this mechanism, software does not need to keep track of FIFO fill levels dynamically.

Alternatively, the SSI can generate interrupt requests and control FIFO filling in software if a particular application requires this approach.

19.5.1.6 DMA to FIRI/SIRI

Select DMA channels can be configured, using DMACR, to respond to FIRI/SIRI requests directly. See Section 23.4.1.22, “DMA Control Register (DMACR).” When so configured, the FIRI/SIRI issues DMA requests when its Tx/Rx FIFO is about to underrun/overrun. These DMA requests are enabled as follows:

- SIRI

  - Enable—SCR1[RXDMAEN,TXDMAEN]. See Section 14.4.1.3, “SIRI Control Register 1 (SCR1)).
  - FIFO watermarks—SFCR[RXTL,TXTL]. See Section 14.4.1.7, “SIRI FIFO Control Register (SFCR)).
- FIRI
  - Transmit
    - FIFO watermark—FIRITCR[TDT]. See Section 14.4.1.17, “FIRI Transmitter Control Register (FIRITCR).”
  - Receive
    - Enable—FIRIRCR[RPEDE]. See Section 14.4.1.19, “FIRI Receiver Control Register (FIRIRCR).”
    - FIFO watermark—FIRIRCR[RDT]. See Section 14.4.1.19, “FIRI Receiver Control Register (FIRIRCR).”

The DMA can then be programmed to respond to these DMA requests and transmit the predetermined amount of data needed by the FIRI/SIRI. With this scheme, the customer does not need to keep track of FIFO fill levels dynamically.

Alternatively, the FIRI/SIRI can generate interrupt requests and control the FIFOs with software if a particular application requires this approach.

Chapter 20
PCI Bus Interface

The PCI interface is compatible with the *PCI Local Bus Specification*, Rev. 2.2. It is beyond the scope of this manual to document the intricacies of the PCI bus. This chapter describes the PCI controller (referenced as PCI throughout this chapter) of this device and provides a basic description of PCI bus operations. The specific emphasis is directed at how the integrated processor implements the PCI bus. Designers of systems incorporating PCI devices should refer to the specification for a thorough description of the PCI bus.

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.

20.1 Introduction

The PCI controller acts as a bridge between the PCI interface and the OCeaN switch fabric. Figure 20-1 is a high-level block diagram of the PCI controller.

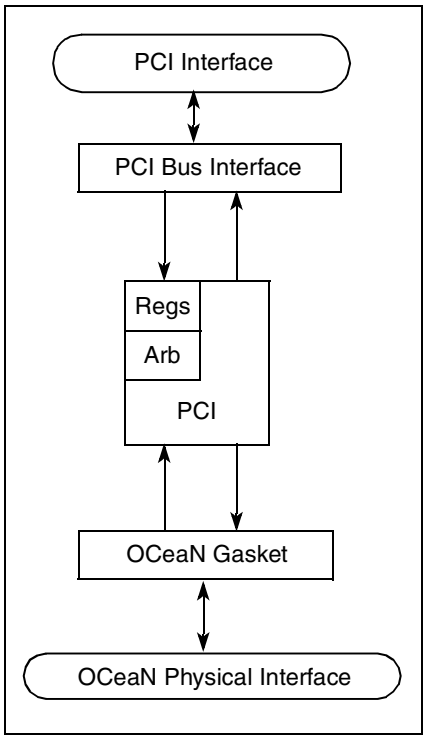


Figure 20-1. PCI Controller Block Diagram

20.1.1 Overview

The PCI controller connects the OCeaN to the PCI bus, to which I/O components are connected. The PCI bus uses a 32-bit multiplexed address/data bus, plus various control and error signals. The PCI interface supports address and data parity with error checking and reporting.

The integrated processor's PCI interface functions both as a master (initiator) and a target device. Internally, the design is divided into the following:

- Data path blocks
- Control logic blocks
- Memory

The data path blocks contain the queues, tables for transaction tracking, and ordering. The control blocks contain control logic and state-machines for buffer control, bus protocol, tag generation, and transaction resizing. The memory blocks are used solely for inbound and outbound data storage. This allows the integrated processor to handle separate PCI transactions simultaneously. For example, consider the case where a burst-write transaction from the integrated processor to another PCI device terminates with a disconnect before finishing the transaction. If another PCI device is granted the PCI bus and requests a burst-read from local memory, the integrated processor, as a target, can accept the burst-read transfer. When the integrated processor is granted mastership of the PCI bus, the burst-write transaction continues. 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.

There are two blocks of memory in the design:

- The inbound buffers
- The outbound read buffers combined with the outbound write buffers

There are many blocks of control logic in the block. On the PCI side there are machines for PCI controller initiated address and data tenures for inbound and outbound data, respectively. On the OCeaN side there are machines for fabric arbitration, outbound data, and inbound data.

As an initiator, the integrated processor supports read and write operations to the PCI memory space, the PCI I/O space, and the 256-byte PCI configuration space. As an initiator, the integrated processor also supports generating PCI special-cycle and interrupt-acknowledge transactions. As a target, the integrated processor supports read and write operations to local memory, and, when configured in agent mode, read and write operations to the internal PCI configuration registers.

The integrated processor can function as either a PCI host bridge (host mode) or a peripheral device on the PCI bus (agent mode). See Section 20.1.3.1.1, “Host Mode,” for more information.

In agent mode, all of the PCI configuration registers in the integrated processor can be programmed from the PCI bus. See Section 20.4.2.11.3, “Agent Accessing the PCI Configuration Space,” for more information.

The PCI interface provides bus arbitration for the integrated processor and up to five other PCI bus masters. The arbitration algorithm is a programmable two-level, round-robin priority selector. The on-chip PCI arbiter can operate in both host and agent modes or it can be disabled to allow for an external PCI arbiter.

The integrated processor also provides an address translation mechanism to map inbound PCI to OCeaN accesses and outbound OCeaN to PCI accesses.

20.1.1.1 Outbound Transactions

Upon detecting an OCeaN-to-PCI transaction, the integrated processor requests the use of the PCI bus. For OCeaN-to-PCI bus write operations, the integrated processor requests mastership of the PCI bus when the source completes the write operation to the OCeaN. For OCeaN-to-PCI read operations, the integrated processor requests mastership of the PCI bus when it decodes that the access is for PCI address space.

Once granted, the integrated processor drives the address (PCI_AD[31:0]) and the bus command (PCI_C/$\overline{\text{BE}}$[3:0]) signals.

The master part of the interface can initiate master-abort cycles, recognizes target-abort, target-retry, and target-disconnect cycles, and supports various device selection timings. The master interface does not run fast back-to-back or exclusive accesses.

20.1.1.2 Inbound Transactions

Upon detection of a PCI address phase, the integrated processor decodes the address and bus command to determine if the transaction is within the local memory access boundaries. If the transaction is destined for

local memory, the target interface latches the address, decodes the PCI bus command, and forwards the transaction to the OCeaN control unit. On writes to local memory, data is forwarded along with the byte enables (if applicable) to the internal control unit. 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 bus interface. On reads, the data is driven on the bus and the byte enables (if applicable) determine which byte lanes contain meaningful data.

The target interface of the integrated processor can issue target-abort, target-retry, and target-disconnect cycles. The target interface supports fast back-to-back transactions. The target interface uses the fastest device selection timing.

The integrated processor supports data streaming to and from local memory. This means that data can flow between the processor PCI interface and local memory as long as the internal buffers are not filled.

20.1.2 Features

The following is a list of PCI features that is supported:

- PCI interface 2.2 compatible
- 66- and 33-MHz support
- 32-bit PCI interface support on PCI port
- Host and agent mode support
- 64-bit dual address cycle (DAC) support
- On-chip arbitration with support for five high-priority request and grant signal pairs
- Support for accesses to all PCI memory and I/O address spaces
- Support for PCI-to-memory and memory-to-PCI streaming
- Memory prefetching of PCI read accesses
- Support posting of processor-to-PCI and PCI-to-memory writes
- Support selectable snoop for inbound accesses
- PCI configuration registers
- PCI 3.3-V compatible

20.1.3 Modes of Operation

A number of parameters that affect the PCI controller modes of operation are determined at power-on reset (POR) by reset configuration signals as described in Chapter 4, “Reset, Clocking, and Initialization.” Table 20-1 provides a summary of these modes.

Table 20-1. POR Parameters for PCI Controller

Parameter	Description	Section/page
Host/agent configuration	Selects between host and agent mode for the PCI interface.	4.4.3.11/4-16
PCI clocking	Selects between asynchronous or synchronous clocking for PCI	4.4.3.14/4-18
PCI arbiter enable	Enables the on-chip PCI bus arbiter	4.4.3.17/4-19
PCI I/O impedance	Selects the impedance of the PCI I/O drivers	4.4.3.16/4-18

20.1.3.1 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). Additionally, the PCI controller can operate in agent configuration lock mode. Note that host/agent mode selection is determined at power-up.

20.1.3.1.1 Host Mode

When the device powers up in host mode, all inbound configuration accesses are ignored (and thus master aborted). See Section 20.5.1.1, “Host Mode,” for more information.

20.1.3.1.2 Agent Mode

When the device powers up in agent mode, it acknowledges inbound configuration accesses. See Section 20.5.1.2, “Agent Mode,” for more information. 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.

20.1.3.1.3 Agent Configuration Lock Mode

When the device powers up in agent configuration lock mode, it retries inbound configuration accesses until the ACL bit in the PCI bus function register is cleared. See Section 20.5.1.3, “Agent Configuration Lock Mode,” for more information.

20.1.3.2 PCI Clocking Configuration

The interface can be configured to be clocked asynchronously with a PCI_CLK input or synchronously with the SYSCLK input. The initial value for clocking is determined by a power-on reset configuration signal.

20.1.3.3 PCI Arbiter (Internal/External Arbiter) Configuration

The interface can be configured to use an on-chip or off-chip PCI arbiter. The initial value for the arbiter is determined by a power-on reset configuration signal.

20.1.3.4 PCI Impedance Configuration

The device has a programmable impedance for PCI bus signals. The initial value for impedance is determined by a power-on reset configuration signal.

20.2 External Signal Descriptions

Figure 20-2 shows the external PCI signals.

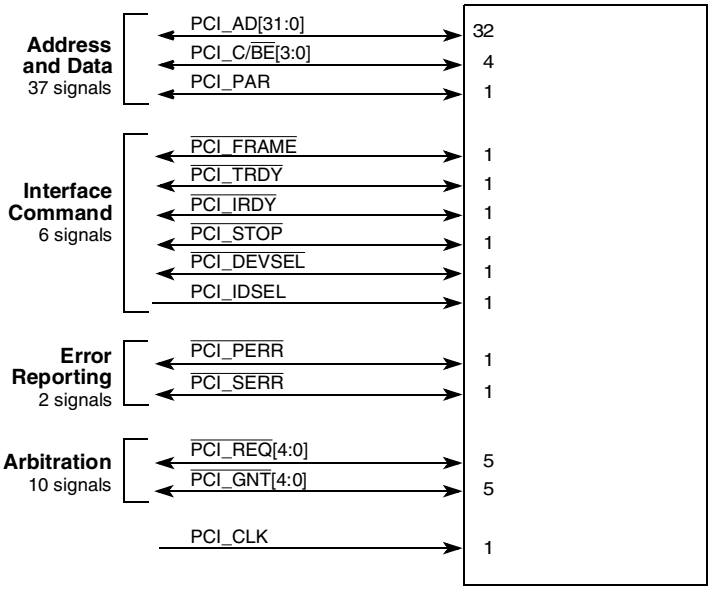


Figure 20-2. PCI Interface External Signals

Table 20-2 contains the detailed descriptions of the external PCI interface signals.

Table 20-2. PCI Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
PCI_AD[31:0]	I/O	PCI address/data bus. The PCI address/data bus consists of signals that are both input and output signals on this PCI controller.	
	O	As outputs for the bidirectional PCI address/data bus, these signals operate as described below.	
		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. The PCI_AD[7:0] signals define the LSB and PCI_AD[31:24] the MSB.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
	I	As inputs for the bidirectional PCI address/data bus, these signals operate as described below.	
		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. The PCI_AD[7:0] signals define the LSB and PCI_AD[31:24] the MSB.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2

Table 20-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_C/$\overline{\text{BE}}$[3:0]	I/O	Command/byte enable. The command/byte enable signals are both input and output signals on this PCI controller. The command encodings for PCI bus mode are described in Section 20.4.2.2, "PCI Bus Commands."	
	O	As outputs for the bidirectional command/byte enable, these signals operate as described below.	
		State Meaning	Asserted/Negated—During the address phase, PCI_C/$\overline{\text{BE}}$[3:0] define the bus command. During the data phase, PCI_C/$\overline{\text{BE}}$[3:0] act as byte enables indicating which byte lanes carry meaningful data. The PCI_C/$\overline{\text{BE}}$[0] signal applies to the LSB.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
	I	As inputs for the bidirectional command/byte enable, these signals operate as described below.	
		State Meaning	Asserted/Negated—During the address phase, PCI_C/$\overline{\text{BE}}$[3:0] indicate the command that another master is sending. During the data phase, PCI_C/$\overline{\text{BE}}$[3:0] indicate which byte lanes are valid.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
$\overline{\text{PCI\_DEVSEL}}$	I/O	Device select. The device select signal is both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional device select, these signals operate as described below.	
		State Meaning	Asserted—Indicates that this PCI controller has decoded the address and is the target of the current access. Negated—Indicates that this 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 2.2
	I	As inputs for the bidirectional device select, these signals operate as described below.	
		State Meaning	Asserted—Indicates that some PCI agent (other than this PCI controller) has decoded its address as the target of the current access. Negated—Indicates that no PCI agent has been selected.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
$\overline{\text{PCI\_FRAME}}$	I/O	Frame. The frame signal is both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional frame, these signals operate as described below.	
		State Meaning	Asserted—Indicates that this PCI controller, acting as a PCI master, is initiating a bus transaction. While $\overline{\text{PCI\_FRAME}}$ is asserted, data transfers may continue. Negated—If $\overline{\text{PCI\_IRDY}}$ is asserted, indicates that the PCI transaction is in the final data phase; if $\overline{\text{PCI\_IRDY}}$ is negated, indicates that the PCI bus is idle.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
	I	As inputs for the bidirectional frame, these signals operate as described below.	
		State Meaning	Asserted—Indicates that another PCI master is initiating a bus transaction. Negated—Indicates that 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 2.2

Table 20-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_GNT[4:0]	O	PCI bus grant. Output signals on this PCI controller when the arbiter is enabled. When the arbiter is disabled PCI_GNT0 is an input signal. Note that PCI_GNT[*n*] is a point-to-point signal. Every master has its own bus grant signal. **Note:** PCI_GNT[4:1] are multiplexed with GPIO1 signals. See Section 3.2.1, "PCI Arbitration and GPIO1 Signal Multiplexing," for more information. The functionality of these signals is determined by the general-purpose I/O control register (GPIOCR) in the global utilities block. Note that the GPIOCR defaults to GPIO functionality on the PCI arbitration signal pins; the GPIOCR must be initialized to the desired PCI signaling for proper operation. **Note:** These signals are also used as reset configuration signals as described in Section 4.4.3, "Power-On Reset Configuration."	
		State Meaning	Asserted—Indicates that this PCI controller has granted control of the PCI bus to agent *n*. Negated—Indicates that this PCI controller has not granted control of the PCI bus to agent *n*.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
PCI_IDSEL	I	Initialization device select. The initialization device select signal is an input signal on this PCI controller. It is used as a chip select during configuration read and write transactions.	
		State Meaning	Asserted—Indicates this PCI controller is being selected as a target of a configuration read or write transactions. Negated—Indicates this 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 2.2
PCI_IRDY	I/O	Initiator ready. The initiator ready signal is both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional initiator ready, these signals operate as described below.	
		State Meaning	Asserted—Indicates that this 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 the data bus. During a read, this PCI controller asserts PCI_IRDY to indicate that it is prepared to accept data. Negated—Indicates that 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 2.2
	I	As inputs for the bidirectional initiator ready, these signals operate as described below.	
		State Meaning	Asserted—Indicates another PCI master is able to 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 the PCI bus is idle.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev

Table 20-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_PAR	I/O	PCI parity. The PCI parity signal is both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional PCI parity, these signals operate as described below.	
		State Meaning	Asserted—Indicates odd parity across the PCI_AD[31:0] and PCI_C/$\overline{\text{BE}}$[3:0] signals during address and data phases. Negated—Indicates even parity across the PCI_AD[31:0] and PCI_C/$\overline{\text{BE}}$[3:0] signals during address and data phases.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
	I	As inputs for the bidirectional PCI parity, these signals operate as described below.	
		State Meaning	Asserted—Indicates odd parity driven by another PCI master or the PCI target during read data phases. Negated—Indicates even parity driven by another PCI master or the PCI target during read data phases.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
$\overline{\text{PCI\_PERR}}$	I/O	PCI parity error. The PCI parity error signal is both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional PCI parity error, these signals operate as described below.	
		State Meaning	Asserted—Indicates that this PCI controller, acting as a PCI agent, detected a data parity error. (The PCI initiator drives $\overline{\text{PCI\_PERR}}$ on read operations; the PCI target drives $\overline{\text{PCI\_PERR}}$ on write operations.) Negated—Indicates no error.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
	I	As inputs for the bidirectional PCI parity error, these signals operate as described below.	
		State Meaning	Asserted—Indicates that another PCI agent detected a data parity error while this PCI controller was sourcing data (this PCI controller was acting as the PCI initiator during a write, or was acting as the PCI target during a read). Negated—Indicates no error.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
$\overline{\text{PCI\_REQ}}$[4:0]	I	PCI bus request. Input signals on this PCI controller when the arbiter is enabled. When the arbiter is disabled, $\overline{\text{PCI\_REQ}}$[0] is an output. 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 $\overline{\text{PCI\_REQ}}$[*n*] input. **Note:** $\overline{\text{PCI\_REQ}}$[4:1] are multiplexed with GPIO1 signals. See Section 3.2.1, "PCI Arbitration and GPIO1 Signal Multiplexing," for more information. The functionality of these signals is determined by the general-purpose I/O control register (GPIOCR) in the global utilities block. Note that the GPIOCR defaults to GPIO functionality on the PCI arbitration signal pins; the GPIOCR must be initialized to the desired PCI signaling for proper operation.	
		State Meaning	Asserted—Indicates that agent *n* is requesting control of the PCI bus to perform a transaction. Negated—Indicates that agent *n* does not require use of the PCI bus.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2

Table 20-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_SERR	I/O	PCI system error.The PCI system error signal is both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional PCI system error, these signals operate as described below.	
		State Meaning	Asserted—Indicates that 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—Indicates no error.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
	I	As inputs for the bidirectional PCI system error, these signals operate as described below.	
		State Meaning	Asserted—Indicates that a target (other than this PCI controller) has detected a catastrophic error. Negated—Indicates no error.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
PCI_STOP	I/O	Stop.The stop signal is both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional stop, these signals operate as described below.	
		State Meaning	Asserted—Indicates that this PCI controller, acting as a PCI target, is requesting that the initiator stop the current transaction. Negated—Indicates that the current transaction can continue.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
	I	As inputs for the bidirectional stop, these signals operate as described below.	
		State Meaning	Asserted—Indicates that a target is requesting that the PCI initiator stop the current transaction. Negated—Indicates that the current transaction can continue.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2
PCI_TRDY	I/O	Target ready. Both an input and output signal on this PCI controller.	
	O	As outputs for the bidirectional target ready, these signals operate as described below.	
		State Meaning	Asserted—Indicates that this 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 the data bus. During a write, this PCI controller asserts PCI_TRDY to indicate that it is prepared to accept data. Negated—Indicates that 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 2.2
	I	As inputs for the bidirectional target ready, these signals operate as described below.	
		State Meaning	Asserted—Another PCI target is able to complete the current data phase of a transaction. Negated—Indicates a wait cycle from another target.
		Timing	Assertion/Negation—As specified by PCI Local Bus Specification Rev 2.2

Table 20-2. PCI Interface Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
PCI_CLK	I	PCI clock is an independent clock that may be used for the PCI interface. If used the PCI operation is asynchronous with respect to SYSCLK and the platform clock. In order to used this signal as the PCI clock source, it must be designated during POR configuration. See the reset chapter for POR details regarding clock selection as well as proper PCI frequency selection.	
		Timing	Assertion/Negation—See the device *Hardware Specification* for specific timing information.

20.3 Memory Map/Register Definitions

The PCI controller supports the following two types of registers:

- Memory-mapped registers—these registers control PCI address translation, PCI error management, and PCI configuration register access. These registers are described in Section 20.3.1, "PCI Memory-Mapped Registers," and its subsections.
- PCI configuration registers contained within the PCI configuration header—these registers are specified by the PCI bus specification for every PCI device. These registers are described in Section 20.3.2, "PCI Configuration Header," and its subsections.

20.3.1 PCI Memory-Mapped Registers

The PCI memory mapped registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR on the local side or the PCSRBAR on the PCI side) plus the block base address, plus the offset of the specific register to be accessed. Note that all memory-mapped registers (except the PCI configuration data register, PCI CFG_DATA) must only be accessed as 32-bit quantities.

Table 20-3 lists the memory-mapped registers.

Table 20-3. PCI Memory-Mapped Register Map

Offset	Register	Access	Reset	Section/page
PCI Controller Memory-Mapped Registers—Block Base Address 0x0_8000				
PCI Configuration Access Registers				
0x000	CFG_ADDR—PCI configuration address	R/W	All zeros	20.3.1.1.1/20-14
0x004	CFG_DATA—PCI configuration data	R/W	All zeros	20.3.1.1.2/20-15
0x008	INT_ACK—PCI interrupt acknowledge	R	All zeros	20.3.1.1.3/20-15
0x00C–0xBFC	Reserved	—	—	—
PCI ATMU Registers—Outbound and Inbound				
0xC00–0xC3C–Outbound Window 0 (default)				
0xC00	POTAR0—PCI outbound window 0 (default) translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC04	POTEAR0—PCI outbound window 0 (default) translation extended address register	R/W	All zeros	20.3.1.2.2/20-16

Table 20-3. PCI Memory-Mapped Register Map (continued)

Offset	Register	Access	Reset	Section/page
0xC08	Reserved	—	—	
0xC0C	Reserved	—	—	
0xC10	POWAR0—PCI outbound window 0 (default) attributes register	R/W	0x8004_401F	20.3.1.2.4/20-17
0xC14–0xC1C	Reserved	—	—	
0xC20–0xC3C—Outbound Window 1				
0xC20	POTAR1—PCI outbound window 1 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC24	POTEAR1—PCI outbound window 1 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC28	POWBAR1—PCI outbound window 1 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC2C	Reserved	—	—	
0xC30	POWAR1—PCI outbound window 1 attributes register	R/W	All zeros	20.3.1.2.4/20-17
0xC34–0xC3C	Reserved	—	—	
0xC40–0xC5C—Outbound Window 2				
0xC40	POTAR2—PCI outbound window 2 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC44	POTEAR2—PCI outbound window 2 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC48	POWBAR2—PCI outbound window 2 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC4C	Reserved	—	—	
0xC50	POWAR2—PCI outbound window 2 attributes register	R/W	All zeros	20.3.1.2.4/20-17
0xC54–0xC5C	Reserved	—	—	
0xC60–0xC7C—Outbound Window 3				
0xC60	POTAR3—PCI outbound window 3 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC64	POTEAR3—PCI outbound window 3 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC68	POWBAR3—PCI outbound window 3 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC6C	Reserved	—	—	
0xC70	POWAR3—PCI outbound window 3 attributes register	R/W	All zeros	20.3.1.2.4/20-17
0xC74–0xC7C	Reserved	—	—	
0xC80–0xC9C—Outbound Window 4				
0xC80	POTAR4—PCI outbound window 4 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC84	POTEAR4—PCI outbound window 4 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC88	POWBAR4—PCI outbound window 4 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC8C	Reserved	—	—	
0xC90	POWAR4—PCI outbound window 4 attributes register	R/W	All zeros	20.3.1.2.4/20-17

Table 20-3. PCI Memory-Mapped Register Map (continued)

Offset	Register	Access	Reset	Section/page
0xC94–0xD9C	Reserved	—	—	
0xDA0–0xDBC–Inbound Window 3				
0xDA0	PITAR3—PCI inbound window 3 translation address register	R/W	All zeros	20.3.1.3.1/20-20
0xDA4	Reserved	—	—	
0xDA8	PIWBAR3—PCI inbound window 3 base address register	R/W	All zeros	20.3.1.3.2/20-20
0xDAC	PIWBEAR3—PCI inbound window 3 base extended address register	R/W	All zeros	20.3.1.3.3/20-21
0xDB0	PIWAR3—PCI inbound window 3 attributes register	R/W	All zeros	20.3.1.3.4/20-21
0xDB4–0xDBC	Reserved	—	—	
0xDC0–0xDDC–Inbound Window 2				
0xDC0	PITAR2—PCI inbound window 2 translation address register	R/W	All zeros	20.3.1.3.1/20-20
0xDC4	Reserved	—	—	
0xDC8	PIWBAR2—PCI inbound window 2 base address register	R/W	All zeros	20.3.1.3.2/20-20
0xDCC	PIWBEAR2—PCI inbound window 2 base extended address register	R/W	All zeros	20.3.1.3.3/20-21
0xDD0	PIWAR2—PCI inbound window 2 attributes register	R/W	All zeros	20.3.1.3.4/20-21
0xDD4–0xDDC	Reserved	—	—	
0xDE0–0xDFC–Inbound Window 1				
0xDE0	PITAR1—PCI inbound window 1 translation address register	R/W	All zeros	20.3.1.3.1/20-20
0xDE4	Reserved	—	—	
0xDE8	PIWBAR1—PCI inbound window 1 base address register	R/W	All zeros	20.3.1.3.2/20-20
0xDEC	Reserved	—	—	
0xDF0	PIWAR1—PCI inbound window 1 attributes register	R/W	All zeros	20.3.1.3.4/20-21
0xDF4–0xDFC	Reserved	—	—	
PCI Error Management Registers				
0xE00	ERR_DR—PCI error detect register	w1c	All zeros	20.3.1.4.1/20-24
0xE04	ERR_CAP_DR—PCI error capture disabled register	R/W	All zeros	20.3.1.4.2/20-25
0xE08	ERR_EN—PCI error enable register	R/W	All zeros	20.3.1.4.3/20-26
0xE0C	ERR_ATTRIB—PCI error attributes capture register	R/W	All zeros	20.3.1.4.4/20-27
0xE10	ERR_ADDR—PCI error address capture register	R/W	All zeros	20.3.1.4.5/20-28
0xE14	ERR_EXT_ADDR—PCI error extended address capture register	R/W	All zeros	20.3.1.4.6/20-28
0xE18	ERR_DL—PCI error data low capture register	R/W	All zeros	20.3.1.4.7/20-29
0xE1C	ERR_DH—PCI error data high capture register	R/W	All zeros	20.3.1.4.8/20-29
0xE20	GAS_TIMR—PCI gasket timer register	R/W	0x0100_3FFF	20.3.1.4.9/20-29

Table 20-3. PCI Memory-Mapped Register Map (continued)

Offset	Register	Access	Reset	Section/page
0xE28–0xEFC	Reserved	—	—	
0xF00–0xFFC	Reserved for debug	—	—	

20.3.1.1 PCI Configuration Access Registers

The PCI configuration header, shown in Figure 20-24 and Figure 20-58, is accessed through an indirect method utilizing a pair of 32-bit memory-mapped access registers. For PCI, CFG_ADDR is at offset 0x000 and CFG_DATA is at offset 0x004.

20.3.1.1.1 PCI Configuration Address Register (CFG_ADDR)

The CFG_ADDR register is shown in Figure 20-3.

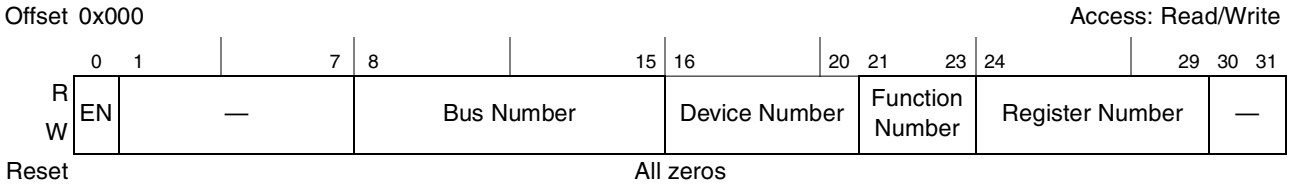


Figure 20-3. PCI CFG_ADDR Register

Table 20-4 describes the bit settings for the CFG_ADDR register.

Table 20-4. PCI CFG_ADDR Field Descriptions

Bits	Name	Description
0	Enable	Allow a PCI configuration access when PCI CFG_DATA is accessed
1–7	—	Reserved
8–15	Bus Number	PCI bus number to access
16–20	Device Number	Device number to access on specified bus
21–23	Function Number	Function to access within specified device
24–29	Register Number	32-bit register to access within specified device
30–31	—	Reserved, hardwired to logic 00

Bus number 0xb00 and device number 0b0_0000 are used to configure the internal PCI configuration header of the PCI controller itself.

See Section 20.4.2.11.2, “Host Accessing the PCI Configuration Space,” and Section 20.4.2.11.3, “Agent Accessing the PCI Configuration Space,” for usage of PCI CFG_ADDR.

20.3.1.1.2 PCI Configuration Data Register (CFG_DATA)

The CFG_DATA register is shown in Figure 20-3.

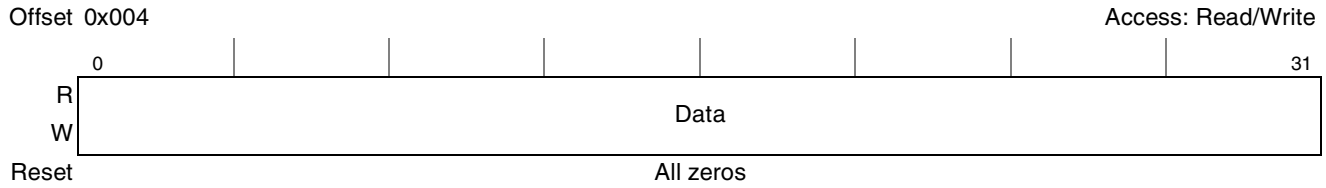


Figure 20-4. PCI CFG_DATA Register

Table 20-5 describes the bit settings for the CFG_DATA register

Table 20-5. PCI CFG_DATA Field Descriptions

Bits	Name	Description
0–31	Data	A read or write to this register starts a PCI configuration cycle if the PCI CFG_ADDR enable bit is set. If the enable bit is not set, a PCI I/O transaction is generated.

The CFG_DATA register is a 4-byte window into the little-endian PCI configuration header data structure; therefore, byte addressing within the CFG_DATA register uses little-endian convention. Note that CFG_DATA may contain 1, 2, 3, or 4 bytes depending on the size of the register being accessed.

See Section 20.4.2.11.2, “Host Accessing the PCI Configuration Space,” and Section 20.4.2.11.3, “Agent Accessing the PCI Configuration Space,” for usage of CFG_DATA.

20.3.1.1.3 PCI Interrupt Acknowledge Register (INT_ACK)

An external PCI interrupt acknowledge transaction is generated by reading the INT_ACK register. For PCI, INT_ACK is at offset 0x008. INT_ACK is shown in Figure 20-5.

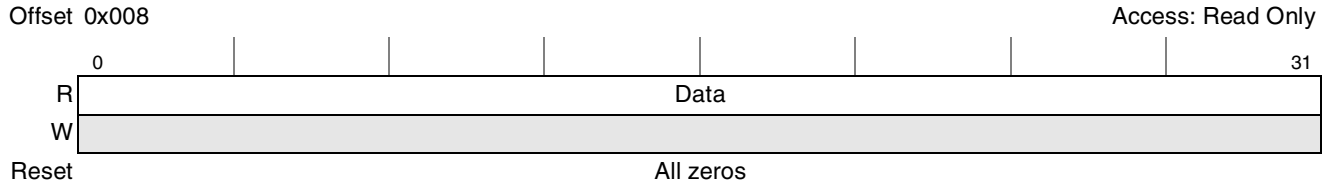


Figure 20-5. PCI INT_ACK Register

Table 20-6 describes the bit settings for the INT_ACK register.

Table 20-6. PCI INT_ACK Field Descriptions

Bits	Name	Description
0–31	Data	A read to this register generates a PCI interrupt acknowledge cycle.

20.3.1.2 PCI ATMU Outbound Registers

The outbound address translation and mapping unit controls the mapping of transactions from the internal platform address space to the external PCI address space. The outbound ATMU consists of four translation windows plus a default translation for transactions that do not hit in one of the four windows.

Each window contains a base address that points to the beginning of the window in the local address map, a translation address that specifies the high-order bits of the transaction in the external PCI address space, and a set of attributes including window size and external transaction type.

Each window must be aligned based on the granularity specified by the window size. If two outbound ATMU windows overlap in the local address space, the mapping of the lower numbered window has precedence over the higher numbered window.

Window 0 is the default window and is the only window enabled upon reset. The default outbound register set is used when a transaction misses in all of the other outbound windows.

20.3.1.2.1 PCI Outbound Translation Address Registers (POTAR*n*)

The PCI outbound translation address registers (POTAR*n*) select the starting addresses in the PCI address space for hits in the PCI outbound windows. The translated address is created by concatenating the transaction offset to this translation address. The format of the POTAR*n* is shown in Figure 20-6.

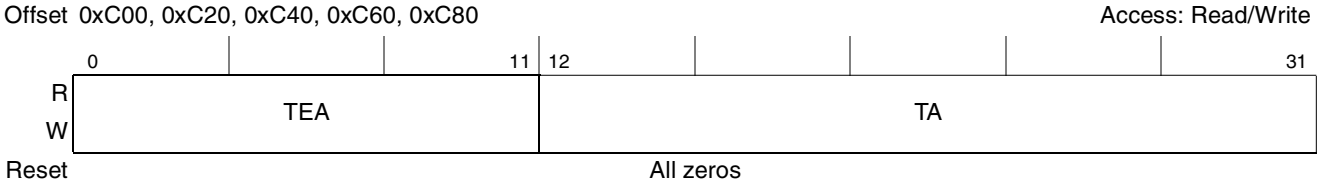


Figure 20-6. PCI Outbound Translation Address Registers (POTAR*n*)

Table 20-7 describes the fields of the POTAR*n* registers.

Table 20-7. POTAR*n* Field Descriptions

Bits	Name	Description
0–11	TEA	Translation extended address. Represents bits [43:32] of a 64-bit PCI address (bit 0 is lsb).
12–31	TA	Translation address. Represents bits [31:12] of the PCI address. The specified address must be aligned to the window size, as defined by POWAR*n*[OWS].

20.3.1.2.2 PCI Outbound Translation Extended Address Registers (POTEAR*n*)

The PCI outbound translation extended address registers (POTEAR*n*) contain the most significant bits of a 64-bit translation address. The format of POTEAR*n* is shown in Figure 20-7.

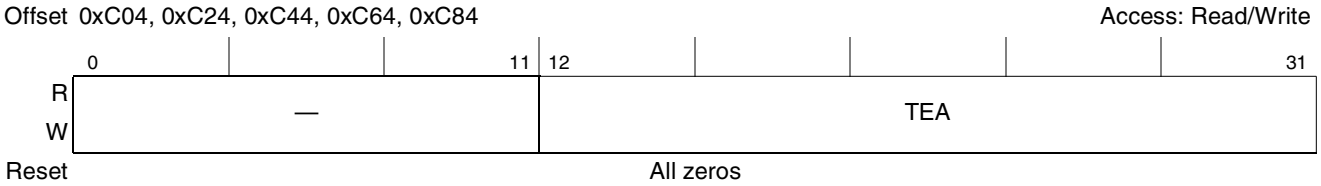


Figure 20-7. PCI Outbound Translation Extended Address Registers (POTEAR*n*)

Table 20-8 describes the fields of the POTEAR*n*.

Table 20-8. POTEAR*n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	TEA	Translation extended address. Comprise bits [63:44] of the translation address.

20.3.1.2.3 PCI Outbound Window Base Address Registers (POWBAR*n*)

The PCI outbound window base address registers (POWBAR*n*) point to the beginning of each translation window in the local 32-bit address space. Addresses for outbound transactions are compared to the appropriate bits in these registers, according to the sizes of the windows. If a transaction does not fall within one of these windows, the default translation and mapping is used. The default window is always enabled and used when the other windows miss.

Note that POWBAR0 (for outbound ATMU window 0) is not used, because window 0 is the default window used when no other windows match. POWBAR0 may be read from and written to, but the value is ignored.

The format of the POWBAR*n* is shown in Figure 20-8.

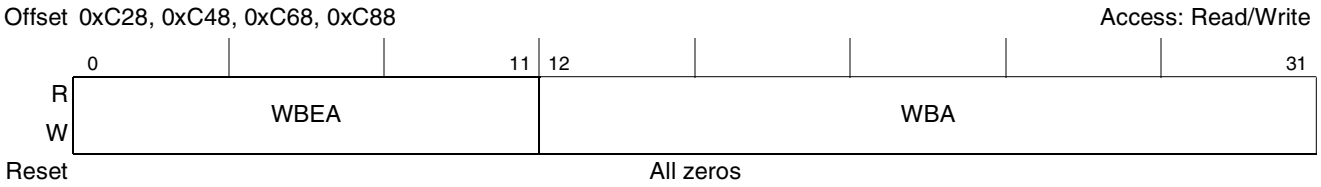


Figure 20-8. PCI Outbound Window Base Address Registers (POWBAR*n*)

Table 20-9 describes the field of the POWBAR*n*.

Table 20-9. POWBAR*n* Field Descriptions

Bits	Name	Description
0–11	WBEA	Window base extended address. Bits 0–7 are reserved; bits 8–11 correspond to bits [0:3] of the (internal platform) base address. 0x000 – 0x00F are valid. 0x010 and greater are reserved.
12–31	WBA	Window base address. Source address which is the starting point for the outbound window. The specified address must be aligned to the window size, as defined by POWAR*n*[OWS]. Corresponds to bits [4-35] of the (internal platform) base address.

20.3.1.2.4 PCI Outbound Window Attributes Registers (POWAR*n*)

The PCI outbound window attributes registers (POWAR*n*) define the window sizes to translate and other attributes for the translations. The minimum window size is 4 Kbytes. The maximum window size is 16 Gbytes.

The default window attribute register, POWAR0, is shown in Figure 20-9. Note that the fields for all of the POWAR*n* registers are the same, only the reset values are different.

Offset 0xC10 Access: Read/Write

	0	1–11	12–15	16–19	20–25	26–31
R/W	EN	—	RTT	WTT	—	OWS
Reset	1	0 0 0 0 0 0 0 0 0 0 0	0 1 0 0	0 1 0 0	0 0 0 0 0 0	0 1 1 1 1 1[1]

Figure 20-9. PCI Outbound Window 0 (Default) Attributes Register (POWAR0)

[1] The default window is enabled, configured for memory read and memory write, and set to an OWS size of 4 Gbytes.

POWAR1–POWAR4 are shown in Figure 20-10.

Offset 0xC30, 0xC50, 0xC70, 0xC90 Access: Read/Write

	0	1–11	12–15	16–19	20–25	26–31
R/W	EN	—	RTT	WTT	—	OWS
Reset	All zeros					

Figure 20-10. PCI Outbound Window 1–4 Attributes Registers (POWAR1–POWAR4)

Table 20-10 describes the fields for the POWAR*n* registers.

Table 20-10. POWAR*n* Field Descriptions

Bits	Name	Description
0	EN	Enable. Enables this address translation
1–11	—	Reserved
12–15	RTT	Read transaction type to run on PCI 0000 Reserved ... 0011 Reserved 0100 Memory Read 0101 Reserved ... 0111 Reserved 1000 I/O Read 1001 Reserved ... 1111 Reserved
16–19	WTT	Write transaction type to run on PCI 0000 Reserved ... 0011 Reserved 0100 Memory Write 0101 Reserved ... 0111 Reserved 1000 I/O Write 1001 Reserved ... 1111 Reserved

Table 20-10. POWAR*n* Field Descriptions (continued)

Bits	Name	Description
20–25	—	Reserved
26–31	OWS	Outbound window size. Outbound translation window size N which is the encoded 2^(N+1) bytes window size. The smallest window size is 4 Kbytes. 000000Reserved ... 0010114-Kbyte window size 0011008-Kbyte window size ... 0111114-Gbyte window size 1000008-Gbyte window size 10000116-Gbyte window size 100010Reserved ... 111111Reserved The default POWAR register (0xC10) has an OWS value of 011111.

20.3.1.3 PCI ATMU Inbound Registers

The inbound address translation and mapping unit controls the mapping of transactions from the external PCI address space to the internal platform address space. The inbound ATMU is comprised of four windows—a configuration window and three general translation windows. The configuration window has higher priority than all other inbound ATMU windows and takes precedence over them if there is an overlap.

Each window contains the following:

- A base address, which points to the beginning of the window in the external PCI address map. The base address of each window is also accessible by PCI configuration transactions as base address registers within the PCI configuration header, as shown in Figure 20-26. The registers may be read or updated equivalently through the ATMU memory map or through PCI configuration transactions to the PCI configuration header.
- A translation address, which specifies the upper order bits of the transaction in the local address space.
- A set of attributes including window size and internal transaction attributes.

Each window's base address and translation address must be aligned to the size of the window. If two general inbound ATMU windows overlap in the external PCI address space, the mappings of the lower numbered window are applied; PCSRBAR takes priority over any overlapping inbound ATMU window. In addition, if inbound ATMU windows are overlapped, the ATMU windows must not map to the same address with different sets of attributes (other than window size).

Note that PCSRBAR in the PCI configuration header acts as a fourth inbound window that translates a 1-Mbyte region of PCI space to the local configuration space pointed to by CCSRBAR. PCSRBAR can be accessed by PCI configuration cycles or by accessing the PCI configuration header through the PCI CFG_ADDR and PCI CFG_DATA registers. See Section 20.3.1.1.1, "PCI Configuration Address Register (CFG_ADDR)," Section 20.3.1.1.2, "PCI Configuration Data Register (CFG_DATA)," and

Section 20.3.2.11, “PCI Base Address Registers.” All accesses to PCSRBAR have an automatic internal byte lane redirection from the little-endian PCI bus to the big-endian CCSRBAR configuration space.

20.3.1.3.1 PCI Inbound Translation Address Registers (PITAR*n*)

The PCI inbound translation address registers (PITAR*n*) points to the beginning of the local address space for the inbound window. The translated address is created by concatenating the transaction offset to this translation address. The format of the PITAR*n* is shown in Figure 20-11.

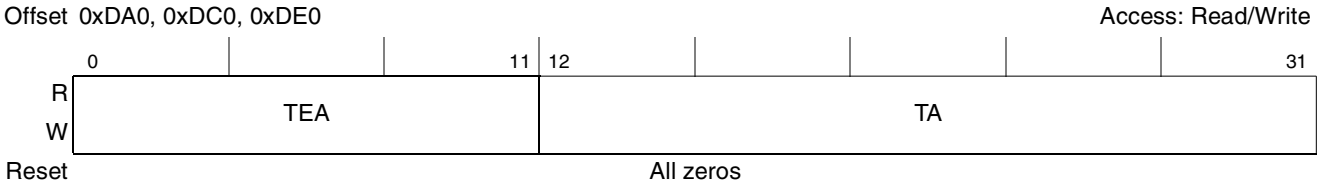


Figure 20-11. PCI Inbound Translation Address Registers (PITAR*n*)

Table 20-11 describes the fields of the PITAR*n* registers

Table 20-11. PITAR*n* Field Descriptions

Bits	Name	Description
0–11	TEA	Translation extended address. Bits 0–7 are reserved; bits 8–11 correspond to bits [0:3] of the local translation address. 0x000 – 0x00F are valid. 0x010 and greater are reserved.
12–31	TA	Translation address. Indicates the starting point of the inbound translated address. The specified address must be aligned to the window size, as defined by PIWAR*n*[IWS]. TA corresponds to bits [4:23] of the 36-bit local translation address.

20.3.1.3.2 PCI Inbound Window Base Address Registers (PIWBAR*n*)

The PCI inbound window base address registers (PIWBAR*n*) select the PCI base address for the windows that are translated to the internal platform address space. Addresses for inbound transactions are compared to these windows. If a PCI transaction does not fall within one of these spaces, then the PCI interface does not assert DEVSEL. The PIWBAR*n* is shown in Figure 20-12.

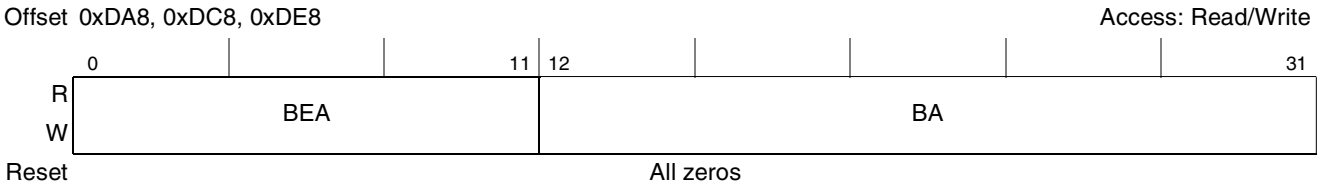


Figure 20-12. PCI Inbound Window Base Address Registers

Table 20-12 describes the fields of the PIWBAR*n* registers.

Table 20-12. PIWBAR Field Descriptions

Bits	Name	Description
0–11	BEA	Base extended address. Corresponds to bits 43–32 of a 64-bit PCI base address.
12–31	BA	Base address. Corresponds to bits 31–12 of a PCI base address. The specified address must be aligned to the window size, as defined by PIWAR*n*[IWS].

20.3.1.3.3 PCI Inbound Window Base Extended Address Registers (PIWBEAR*n*)

The PCI inbound window base extended address registers (PIWBEAR*n*) contain the most-significant bits of a 64-bit base address. Note that inbound window 1 supports only a 32-bit base address and does not define an inbound window base extended address register. The PIWBEAR*n* are shown in Figure 20-13.

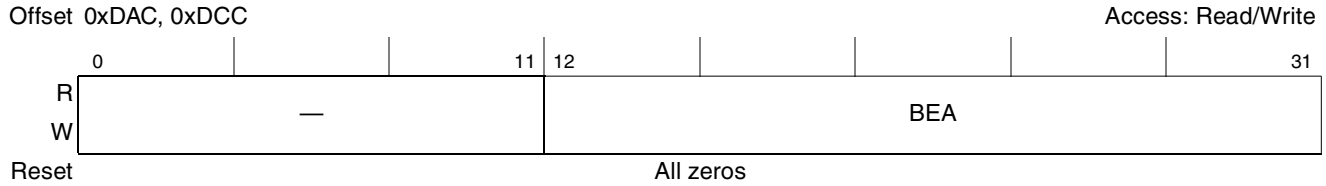


Figure 20-13. PCI Inbound Window Base Extended Address Registers (PIWBEAR*n*)

Table 20-13 describes the fields of the PIWBEAR*n* registers.

Table 20-13. PIWBEAR Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	BEA	Base extended address. Corresponds to bits 63–44 of a 64-bit PCI base address.

20.3.1.3.4 PCI Inbound Window Attributes Registers (PIWAR*n*)

The PCI inbound window attributes registers (PIWAR*n*) define the window sizes to translate and other attributes for the translations. 16 Gbytes is the largest window size allowed. The format of the PIWAR*n* is shown in Figure 20-14.

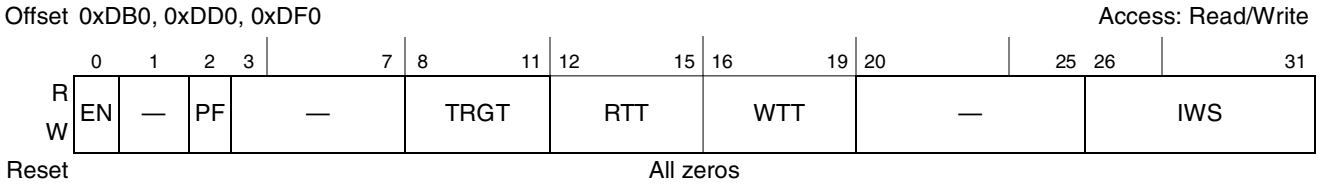


Figure 20-14. PCI Inbound Window Attributes Registers

Table 20-14 describes the fields of the PIWAR*n* registers.

Table 20-14. PIWAR*n* Field Descriptions

Bits	Name	Description
0	EN	Enable. Enables this address translation
1	—	Reserved
2	PF	Prefetchable. Indicates that the address space is prefetchable so that prefetching and streaming are attempted. 0 Not prefetchable 1 Prefetchable
3–7	—	Reserved
8–11	TRGT	Target interface. PIWAR*n*[TRGT] must not be set to target the PCI interface itself. Note that if this field is set to anything other than local address space, the attributes for the transaction must be assigned in a corresponding outbound window at the target. 0000 Reserved 0001 Reserved 0010 PCI Express 1 (×4) on OCN1 0011 Reserved 0100 Reserved 0101 Reserved 0110 Reserved 0111 Reserved 1000 Reserved 1001 Reserved 1010 Reserved 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 Local memory spece (DDR or eLBC)
12–15	RTT	Read transaction type. Transaction type to run if access is a read. The field description differs subject to the transaction being targeted to I/O interface or to local memory. Following are the transaction type settings for reads to an I/O interface: 0000–0011 Reserved 0100 Read 0101–1111Reserved Following are the transaction type settings for reads to local memory: 0000–0011 Reserved 0100 Read, don’t snoop local processor 0101 Read, snoop local processor 0110 Reserved 0111 Read, unlock L2 cache line 1000–1111 Reserved

Table 20-14. PIWAR*n* Field Descriptions (continued)

Bits	Name	Description
16–19	WTT	Write transaction type. Transaction type to run if access is a write. The field description differs subject to the transaction being targeted to an I/O interface or to local memory. Following are the transaction type settings for writes to an I/O interface: 0000–0011 Reserved 0100 Write 0101–1111 Reserved Following are the transaction type settings for writes to local memory: 0000–Reserved 0100 Write, don't snoop local processor 0101 Write, snoop local processor 0110 Write, allocate L2 cache line 0111 Write, allocate and lock L2 cache line 1000–1111 Reserved
20–25	—	Reserved
26–31	IWS	Inbound window size. Inbound translation window size N which is the encoded 2^(N+1) bytes window size. The smallest window is 4 Kbytes. 000000Reserved ... 0010114-Kbyte window size 0011008-Kbyte window size ... 0111114-Gbyte window size 1000008-Gbyte window size 10000116-Gbyte window size 100010Reserved ... 111111Reserved For configuration and run-time registers, the window size is fixed at 0100111-Mbyte window size For register set 0, the window size is limited to 4 Gbytes or smaller.

20.3.1.4 PCI Error Management Registers

When a PCI error is detected, the appropriate error bit is set in the PCI error detect register. Subsequent errors set the appropriate error bits in the error detection registers, but relevant information (attributes, address, and data) is captured only for the first error. The PCI error detect register is a write-1-to-clear type register. That is, reading from this register occurs normally; however, write operations are different in that the bits can be cleared but not set. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear bit 25 and not affect any other bits in the register, the value 0x0000_0040 is written to the register.

The error bit is set regardless of the state of the corresponding error enable bit in the PCI error enable register. The error enable bits are used to send or block the error reporting to the interrupt mechanism. The interrupt can be cleared by writing 0xFFFF_FFFF to the PCI error detect register.

A master-abort condition during a configuration cycle is not necessarily an error. In this case, if relevant, the master abort error enable can be disabled to prevent the reporting of master-aborts during outbound configuration cycles. Master-aborts during configuration reads return 0xFFFF_FFFF.

For an inbound configuration write transaction with a parity error, the device always updates the register access and generates the error interrupt if the interrupt enabled bit is set.

See Section 20.4.2.13, “PCI Error Functions,” for more detail on error handling.

20.3.1.4.1 PCI Error Detect Register (ERR_DR)

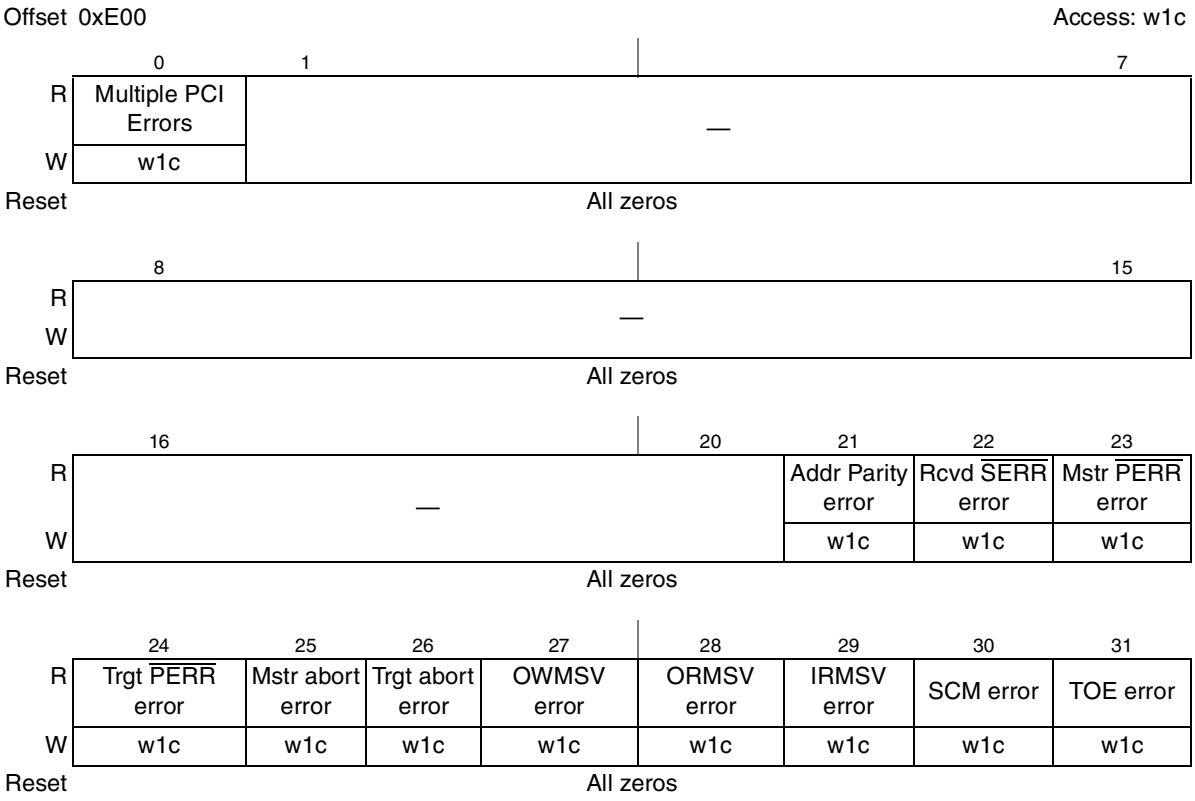


Figure 20-15. PCI Error Detect Register (ERR_DR)

Table 20-15 describes ERR_DR fields. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and an error occurs, the appropriate parity detect and master-abort bits in ERR_DR must be cleared and the appropriate enable bits in ERR_EN must be set to ensure that an interrupt is generated. See Section 6.1.6.2, “Hardware Implementation-Dependent Register 1 (HID1).”

Table 20-15. ERR_DR Field Descriptions

Bits	Name	Description
0	Multiple PCI errors	0 Multiple PCI errors of the same type were not detected (write-1-to-clear) 1 Multiple PCI errors of the same type were detected
1–20	—	Reserved
21	Addr Parity error	Address parity error (write-1-to-clear)
22	Rcvd $\overline{\text{SERR}}$ error	Received $\overline{\text{SERR}}$ error (write-1-to-clear)
23	Mstr $\overline{\text{PERR}}$ error	Master $\overline{\text{PERR}}$ error (write-1-to-clear)

Table 20-15. ERR_DR Field Descriptions (continued)

Bits	Name	Description
24	Trgt $\overline{\text{PERR}}$ error	Target $\overline{\text{PERR}}$ error (write-1-to-clear)
25	Mstr abort error	Master abort error (write-1-to-clear)
26	Trgt abort error	Target abort error (write-1-to-clear)
27	OWMSV error	Outbound write memory space violation error (write-1-to-clear)
28	ORMSV error	Outbound read memory space violation error (write-1-to-clear)
29	IRMSV error	Inbound read memory space violation error (write-1-to-clear)
30	SCM error	Split completion message error (write-1-to-clear)
31	TOE error	Time-out error (write-1-to-clear)

20.3.1.4.2 PCI Error Capture Disable Register (ERR_CAP_DR)

Offset 0xE04 Access: Read/Write

	0	7
R W	—	
Reset	All zeros	

	8	15
R W	—	
Reset	All zeros	

	16	20	21	22	23
R W	—		Addr parity error capture disable	Rcvd $\overline{\text{SERR}}$ error capture disable	Mstr $\overline{\text{PERR}}$ error capture disable
Reset	All zeros				

	24	25	26	27	28	29	30	31
R W	Trgt $\overline{\text{PERR}}$ error capture disable	Mstr abort error capture disable	Trgt abort error capture disable	OWMSV error capture disable	ORMSV error capture disable	IRMSV error capture disable	SCM error capture disable	TOE error capture disable
Reset	All zeros							

Figure 20-16. PCI Error Capture Disable Register (ERR_CAP_DR)

Table 20-16. ERR_CAP_DR Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	Addr parity error capture disable	Disable capture for address parity errors
22	Rcvd $\overline{\text{SERR}}$ error capture disable	Disable capture for received $\overline{\text{SERR}}$ errors

Table 20-16. ERR_CAP_DR Field Descriptions (continued)

Bits	Name	Description
23	Mstr $\overline{\text{PERR}}$ error capture disable	Disable capture for master $\overline{\text{PERR}}$ errors
24	Trgt $\overline{\text{PERR}}$ error capture disable	Disable capture for target $\overline{\text{PERR}}$ errors
25	Mstr abort error capture disable	Disable capture for master abort errors
26	Trgt abort error capture disable	Disable capture for target abort errors
27	OWMSV error capture disable	Disable capture for outbound write memory space violation errors
28	ORMSV error capture disable	Disable capture for outbound read memory space violation errors
29	IRMSV error capture disable	Disable capture for inbound read memory space violation errors
30	SCM error capture disable	Disable capture for split completion message errors
31	TOE error capture disable	Disable capture for time-out errors

20.3.1.4.3 PCI Error Enable Register (ERR_EN)

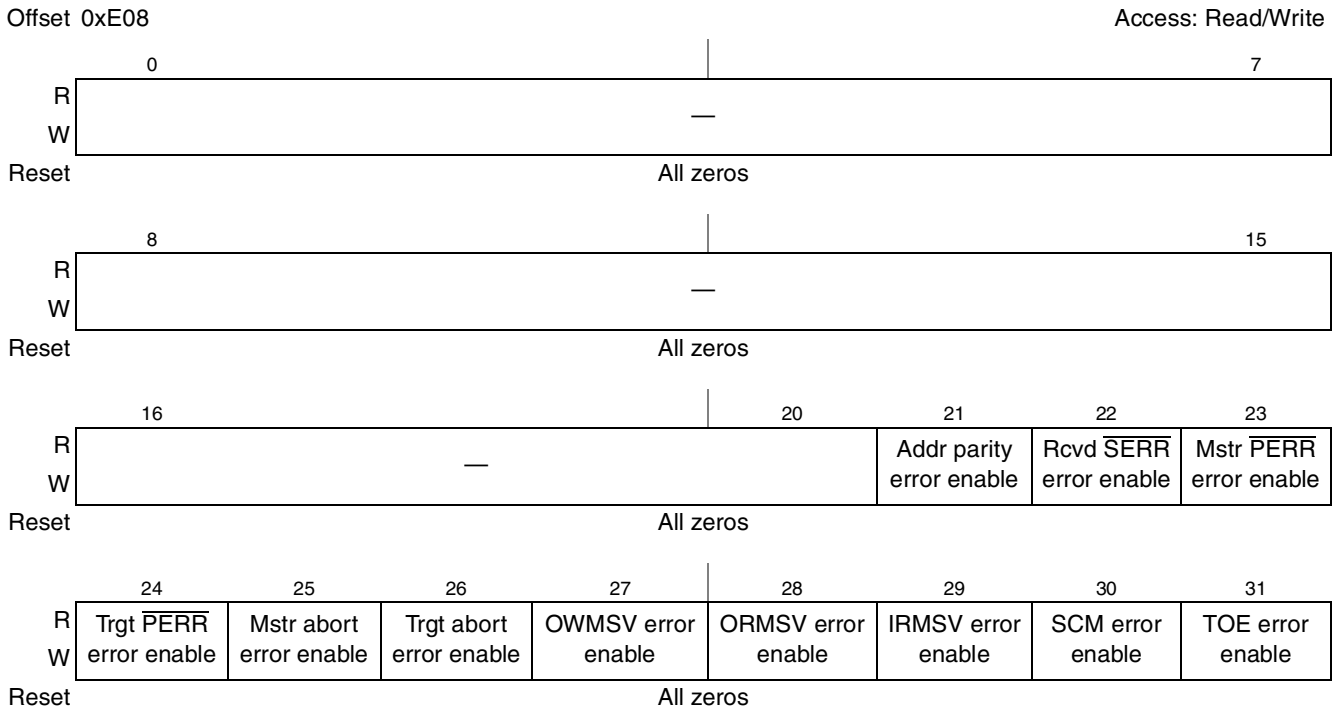


Figure 20-17. PCI Error Enable Register (ERR_EN)

Table 20-17 describes ERR_EN fields. Note that uncorrectable read errors may cause the assertion of *core_fault_in*, which causes the core to generate a machine check interrupt, unless it is disabled (by clearing HID1[RFXE]). If RFXE is zero and this error occurs, the appropriate parity detect and master-abort bits in ERR_DR must be cleared and the appropriate enable bits in ERR_EN must be set to ensure that an interrupt is generated. For more information, see Section 6.1.6.2, "Hardware Implementation-Dependent Register 1 (HID1)."

Table 20-17. ERR_EN Field Descriptions

Bits	Name	Description
0–20	—	Reserved
21	Addr parity error enable	Enable reporting address parity errors
22	Rcvd $\overline{\text{SERR}}$ error enable	Enable reporting received $\overline{\text{SERR}}$ errors
23	Mstr $\overline{\text{PERR}}$ error enable	Enable reporting master $\overline{\text{PERR}}$ errors
24	Trgt $\overline{\text{PERR}}$ error enable	Enable reporting target $\overline{\text{PERR}}$ errors
25	Mstr abort error enable	Enable reporting master abort errors
26	Trgt abort error enable	Enable reporting target abort errors
27	OWMSV error enable	Enable reporting outbound write memory space violation errors
28	ORMSV error enable	Enable reporting outbound read memory space violation errors
29	IRMSV error enable	Enable reporting inbound read memory space violation errors
30	SCM error enable	Enable reporting split completion message errors
31	TOE error enable	Enable reporting time-out errors

20.3.1.4.4 PCI Error Attributes Capture Register (ERR_ATTRIB)

Offset 0xE0C — Access: Read/Write

	0–3	4–7	8	9	10	11–15	16–19	20–30	31
R W	High word byte enables	Low word byte enables	High parity bit	Low parity bit	—	Error source	Command	—	Valid
Reset	All zeros								

Figure 20-18. PCI Error Attributes Capture Register (ERR_ATTRIB)

Table 20-18. ERR_ATTRIB Field Descriptions

Bits	Name	Description
0–3	High word byte enables	PCI byte enables for most significant word of the double word
4–7	Low word byte enables	PCI byte enables for least significant word of the double word
8	High parity bit	Parity bit for most significant PCI bus data word (only valid for 64-bit PCI bus)
9	Low parity bit	Parity bit for least significant PCI bus data word
10	—	Reserved

Table 20-18. ERR_ATTRIB Field Descriptions (continued)

Bits	Name	Description
11–15	Error source	The source of the PCI transaction 00000 PCI 00001 Reserved 00010 PCI Express 1 00011 PCI Express 2 00100 Local bus controller 00101 Reserved 00110 Reserved 00111 Reserved 01000 Configuration space 01001 LCD controller 01010 Boot sequencer 01011 Reserved 01100 Reserved 01101 Reserved 01110 Reserved 01111 DDR memory controller 10000 Reserved 10001 Reserved 10010 Reserved 10011 Reserved 10100 Reserved 10101 DMA1 10110 DMA2 10111 SAP (System access port) 11000 Reserved 11001 Reserved 11010 Reserved 11011 Reserved 11100 Reserved 11101 Reserved 11110 Reserved 11111 Reserved
16–19	Command	PCI command
20–30	—	Reserved
31	Valid info	The PCI bus capture registers contain valid information

20.3.1.4.5 PCI Error Address Capture Register (ERR_ADDR)

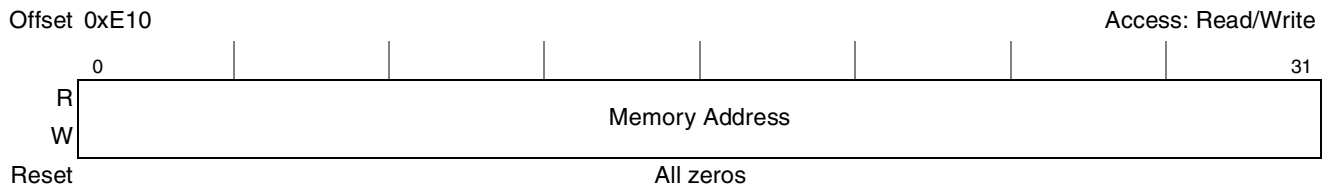


Figure 20-19. PCI Error Address Capture Register (ERR_ADDR)

Table 20-19. ERR_ADDR Field Descriptions

Bits	Name	Description
0–31	Memory address	Memory transaction address

20.3.1.4.6 PCI Error Extended Address Capture Register (ERR_EXT_ADDR)

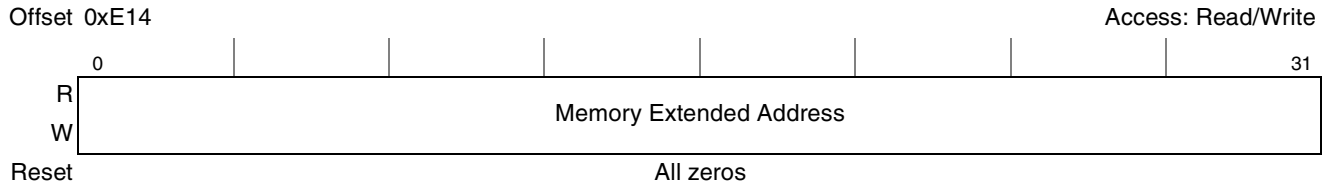


Figure 20-20. PCI Error Extended Address Capture Register (ERR_EXT_ADDR)

Table 20-20. ERR_EXT_ADDR Field Descriptions

Bits	Name	Description
0–31	Memory extended address	Memory transaction extended address

20.3.1.4.7 PCI Error Data Low Capture Register (ERR_DL)

Figure 20-21. PCI Error Data Low Capture Register (ERR_DL)

Table 20-21. ERR_DL Field Description

Bits	Name	Description
0–31	Data low	Least significant PCI bus data word

20.3.1.4.8 PCI Error Data High Capture Register (ERR_DH)

Figure 20-22. PCI Error Data High Capture Register (ERR_DH)

Table 20-22. ERR_DH Field Description

Bits	Name	Description
0–31	Data high	Most significant PCI bus data word (only valid with 64-bit PCI bus)

20.3.1.4.9 PCI Gasket Timer Register (GAS_TIMR)

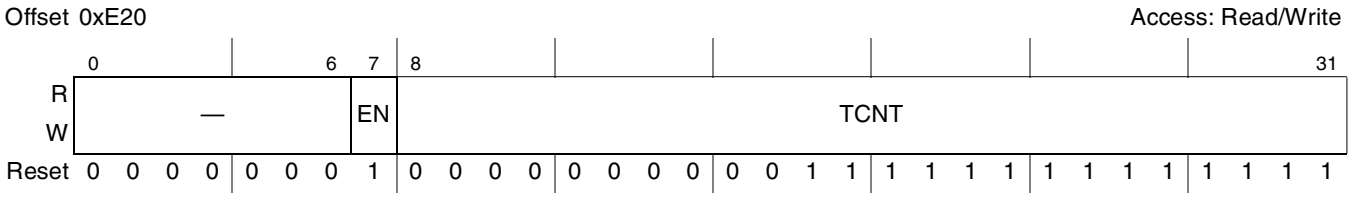


Figure 20-23. PCI Gasket Timer Register (GAS_TIMR)

Table 20-23. GAS_TIMR Field Descriptions

Bits	Name	Description
0–6	—	Reserved
7	EN	Gasket timer enable. 0 gasket timer is disabled. 1 gasket timer is enabled.
8–31	TCNT	Number of system clocks to purge a non-prefetchable inbound read buffer

20.3.2 PCI Configuration Header

The *PCI Local Bus Specification* defines the configuration registers contained within the PCI configuration header from 0x00 through 0x3F. Figure 20-24 lists the common PCI configuration header as implemented by the device.

Reserved

				Address Offset (Hex)
Device ID		Vendor ID		00
PCI Bus Status		PCI Bus Command		04
Bus Base Class Code	Subclass Code	Bus Programming Interface	Revision ID	08
BIST Control	Header Type	Bus Latency Timer	Bus Cache Line Size	0C
PCI Configuration and Status Register Base Address Register (PCSRBAR)				10
32-Bit Memory Base Address Register				14
64-Bit Low Memory Base Address Register				18
64-Bit High Memory Base Address Register				1C
64-Bit Low Memory Base Address Register				20
64-Bit High Memory Base Address Register				24
				28
Subsystem ID		Subsystem Vendor ID		2C
				30
			PCI Bus Capability Pointer	34
				38
PCI Bus MAX_LAT	PCI Bus MIN_GNT	PCI Bus Interrupt Pin	PCI Bus Interrupt Line	3C
				40
PCI Bus Arbiter Configuration		PCI Bus Function		44

Figure 20-24. Common PCI Configuration Header

Table 20-49 in Section 20.4.2.11.1, "PCI Configuration Space Header," provides a summary of the PCI configuration header registers. Detailed descriptions of these registers are provided in the *PCI Local Bus Specification*.

20.3.2.1 PCI Vendor ID Register—Offset 0x00

The PCI vendor ID register, shown in Figure 20-25, is used to identify the manufacturer of the part.

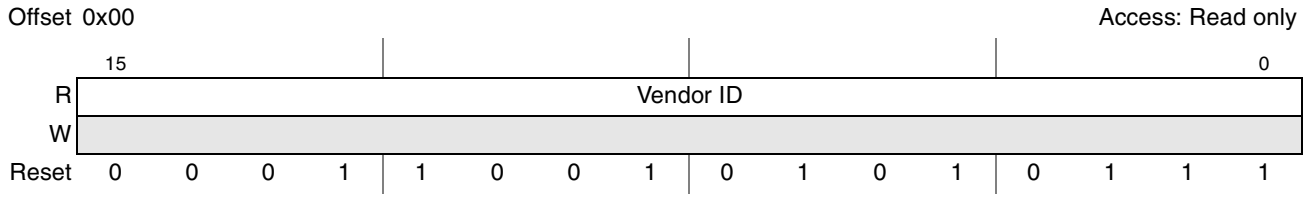


Figure 20-25. PCI Vendor ID Register

Table 20-24 describes PCI vendor ID register fields.

Table 20-24. PCI Vendor ID Register Field Description

Bits	Name	Description
15–0	Vendor ID	0x1957 (Freescale)

20.3.2.2 PCI Device ID Register—Offset 0x02

The PCI device ID register, shown in Figure 20-26, is used to identify the device.

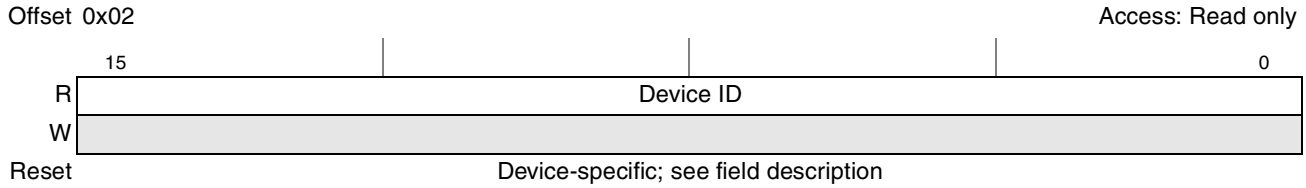


Figure 20-26. PCI Device ID Register

Table 20-25. PCI Device ID Register Field Description

Bits	Name	Description
15–0	Device ID	0x7018MPC8610

20.3.2.3 PCI Bus Command Register—Offset 0x04

The 2-byte PCI bus command register provides control over the ability to generate and respond to PCI cycles. Table 20-26 describes the bits of the PCI bus command register.

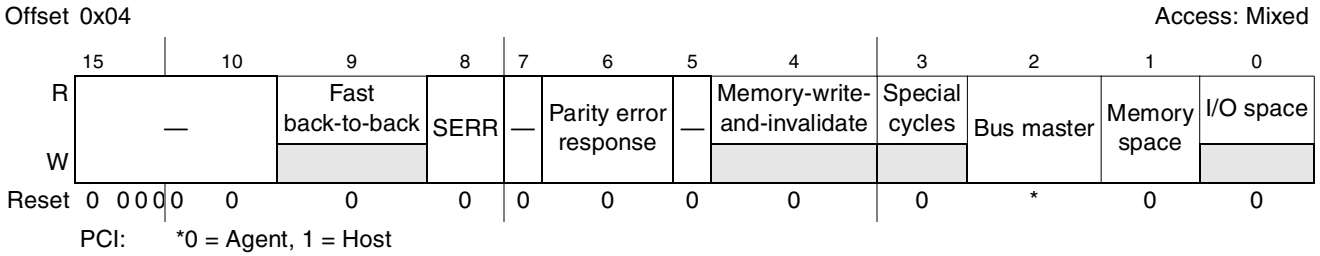


Figure 20-27. PCI Bus Command Register

Table 20-26. PCI Bus Command Register Field Descriptions

Bits	Name	Description
15–10	—	Reserved
9	Fast back-to-back	Hard-wired to 0, indicating that this PCI controller (as a master) does not run fast back-to-back transactions.
8	SERR	Controls the $\overline{\text{PCI\_SERR}}$ driver of this PCI controller. This bit (and bit 6) must be set to report address parity errors. 0 Disables the $\overline{\text{PCI\_SERR}}$ driver 1 Enables the $\overline{\text{PCI\_SERR}}$ driver
7	—	Reserved
6	Parity error response	Controls whether this PCI controller responds to parity errors. 0 Parity errors are ignored and normal operation continues. 1 Parity errors cause the appropriate bit in the PCI status register to be set. However, note that errors are reported based on the values set in the PCI error enable and detection registers.
5	—	Reserved
4	Memory-write-and-invalidate	Hard-wired to 0, indicating that this PCI controller, acting as a master, cannot generate the memory-write-and-invalidate command.
3	Special-cycles	Hard-wired to 0, indicating that this PCI controller (as a target) ignores all special-cycle commands.
2	Bus master	Indicates whether this PCI controller is configured as a master. This indicates the setting of the host/agent configuration input signal at power-on reset. 0 Disables the ability to generate PCI accesses 1 Enables this PCI controller to behave as a PCI bus master (Host)
1	Memory space	Controls whether this PCI controller (as a target) responds to memory accesses. 0 This PCI controller does not respond to PCI memory space accesses. 1 This PCI controller (as a target) responds to PCI memory space accesses.
0	I/O space	Hard-wired to 0, indicating that this PCI controller (as a target) does not respond to PCI I/O space accesses.

20.3.2.4 PCI Bus Status Register—Offset 0x06

The 2-byte PCI bus status register is used to record status information for PCI bus bus-related events. The definition of each bit is given in Table 20-27. Only 2-byte accesses to address offset 0x06 are allowed.

Reads to this register behave normally. Writes are slightly different in that bits can be cleared, but not set. A bit is cleared whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear bit 14 without affecting any other bits in the register, write the value 0b0100_0000_0000_0000 to the register.

Offset 0x06 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	DEVSEL timing		Master data parity error detected
W	w1c	w1c	w1c	w1c	w1c			w1c
Reset	All zeros							

	7	6	5	4	3			0
R	Fast back-to-back capable	—	66-MHz capable	Capabilities list	—			
W								
Reset	1	0	1	0	0	0	0	0

Figure 20-28. PCI Bus Status Register

Table 20-27. PCI Bus Status Register Field Descriptions

Bits	Name	Description
15	Detected parity error	Set whenever this PCI controller detects a PCI parity error, even if parity error handling is disabled (as controlled by bit 6 in the PCI bus command register).
14	Signaled system error	Set whenever this PCI controller asserts $\overline{\text{PCI\_SERR}}$.
13	Received master-abort	Set whenever this PCI controller, acting as the PCI master, terminates a transaction (except for a special-cycle) using master-abort.
12	Received target-abort	Set whenever a PCI transaction initiated by this PCI controller (excluding a special-cycle) is terminated by a target-abort.
11	Signaled target-abort	Set whenever this PCI controller, acting as the PCI target, issues a target-abort to a PCI master.
10–9	DEVSEL timing	Hard-wired to 0b00, indicating that this PCI controller uses fast device select timing.
8	Master data parity error detected	Set upon detecting a data parity error. Three conditions must be met for this bit to be set: • This PCI controller detected a parity error. • This PCI controller was acting as the bus master for the operation in which the error occurred. • Bit 6 in the PCI bus command register was set.
7	Fast back-to-back capable	Hard-wired to 1, indicating that this PCI controller (as a target) is capable of accepting fast back-to-back transactions.
6	—	Reserved
5	66-MHz capable	Read-only bit indicates that this PCI controller is capable of 66 MHz PCI bus operation.
4	Capabilities List	Hard-wired to 0
3–0	—	Reserved

20.3.2.5 PCI Revision ID Register—Offset 0x08

The PCI revision ID register is used to identify the revision of the part.

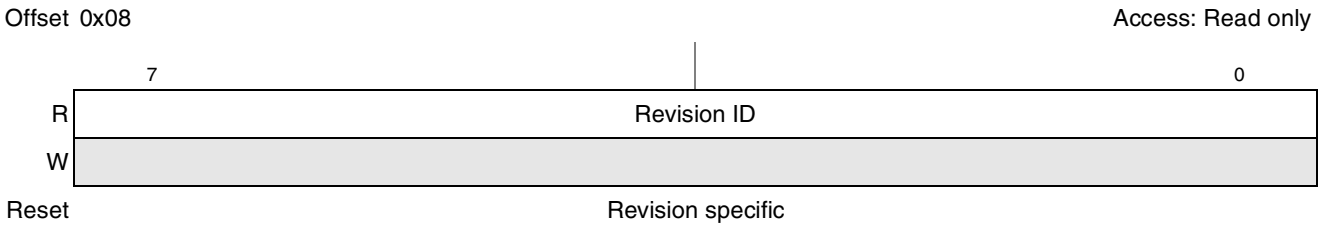


Figure 20-29. PCI Revision ID Register

Table 20-28. PCI Revision ID Register Field Descriptions

Bits	Name	Description
7–0	Revision ID	Revision specific

20.3.2.6 PCI Bus Programming Interface Register—Offset 0x09

Table 20-29 describes the PCI bus programming interface register (PIR).

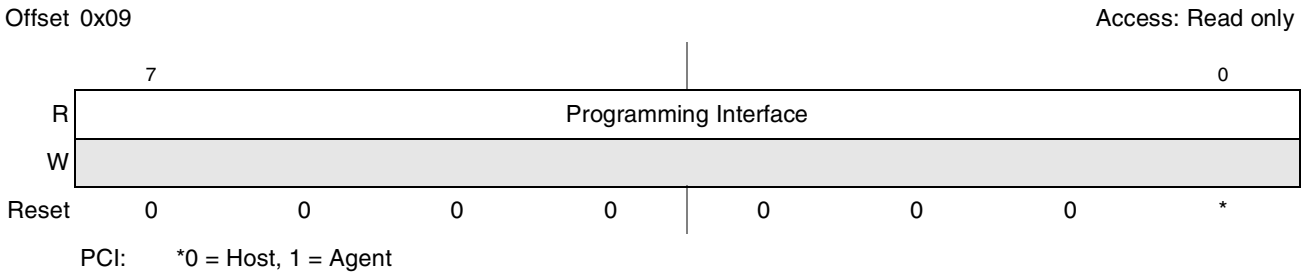


Figure 20-30. PCI Bus Programming Interface Register

Table 20-29. PCI Bus Programming Interface Register Field Description

Bits	Name	Description
7–0	Programming Interface	0x00 When the PCI controller is configured as host bridge 0x01 When the PCI controller is configured as an agent device

20.3.2.7 PCI Subclass Code Register—Offset 0x0A

Table 20-31 describes the PCI subclass code register (PSCR).

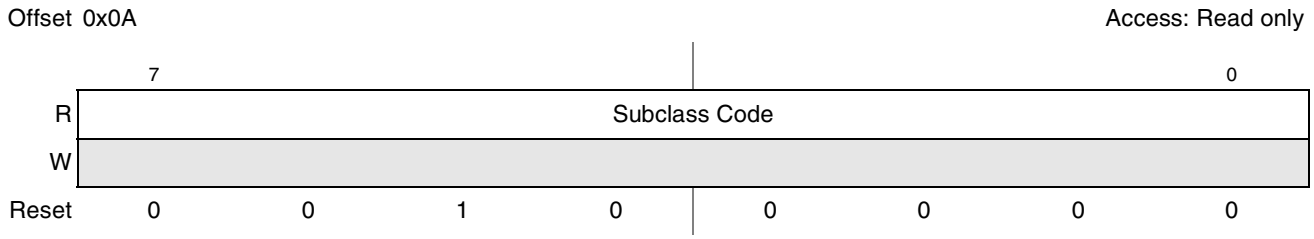


Figure 20-31. PCI Subclass Code Register

Table 20-30. PCI Subclass Code Register Field Description

Bits	Name	Description
7–0	Subclass Code	PowerPC—0x20

20.3.2.8 PCI Bus Base Class Code Register—Offset 0x0B

Table 20-31 describes the PCI bus base class code register (PBCCR).

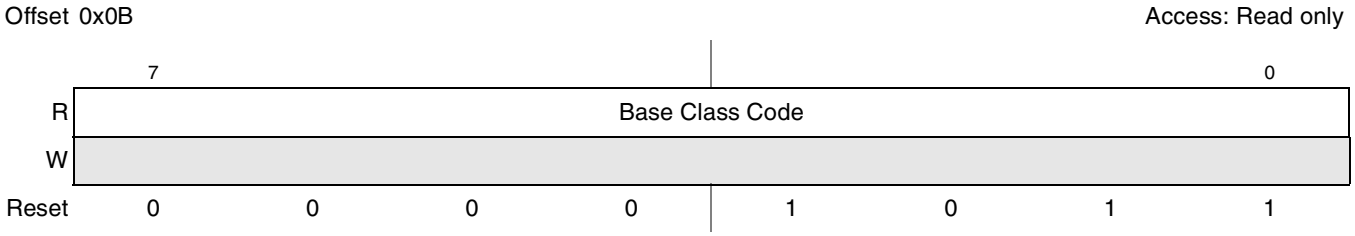


Figure 20-32. PCI Bus Base Class Code Register

Table 20-31. PCI Bus Base Class Code Register Field Description

Bits	Name	Description
7–0	Base Class Code	Processor—0x0B

20.3.2.9 PCI Bus Cache Line Size Register—Offset 0x0C

Table 20-32 describes the PCI bus cache line size register (PCLSR).

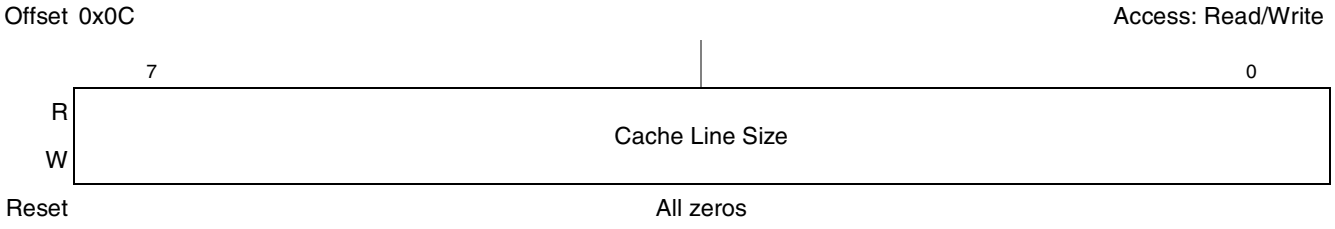


Figure 20-33. PCI Bus Cache Line Size Register

Table 20-32. PCI Bus Cache Line Size Register Field Descriptions

Bits	Name	Description
7–0	Cache Line Size	Represents the cache line size of the processor in terms of 32-bit words (8 32-bit words = 32 bytes). PCLSR is read-write; however, for PCI operation an attempt to program this register to any value other than 0x8 results in clearing it.

20.3.2.10 PCI Bus Latency Timer Register—0x0D

Table 20-33 describes the PCI latency timer register (PLTR).

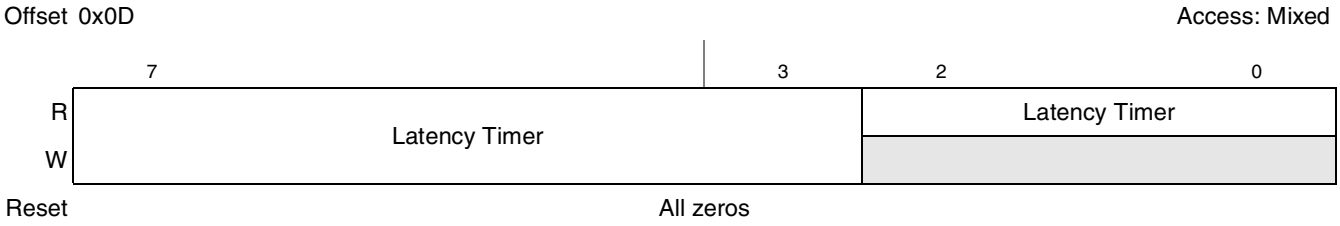


Figure 20-34. PCI Bus Latency Timer Register

Table 20-33. PCI Bus Latency Timer Register Field Descriptions

Bits	Name	Description
7–3	Latency Timer	The maximum number of PCI clocks that the device, when mastering a transaction, holds the bus after PCI bus grant has been negated The value is in PCI clocks. The PCI 2.2 specification gives rules by which the PCI bus interface unit completes transactions when the timer has expired.
2–0	Latency Timer	Read-only bits. The minimum latency timer value when set is 8 PCI clocks.

20.3.2.11 PCI Base Address Registers

A PCI base address register points to the beginnings of each address range to which the device responds by asserting PCI_DEVSEL. The base address register (BAR) at offset 0x10 is a fixed 1-Mbyte window that is automatically translated to the local configuration, control, and status registers address space.

The other base address registers are aliases (with differing format) of the PCI inbound ATMU windows; see Section 20.3.1.3, “PCI ATMU Inbound Registers.” The 32-bit base address register at offset 0x14 corresponds to inbound ATMU window 1; the 64-bit base address registers at offsets 0x18 and 0x20 correspond to inbound ATMU windows 2 and 3. If one of these registers is written, the corresponding ATMU register is also updated; if a PCI inbound ATMU register is written, the corresponding BAR is also updated. If one of these registers is read, the corresponding size of ATMU is returned on the PCI bus providing valid window size in the Inbound ATMU window attributes register.

Note that PCSRBAR cannot be updated through the inbound ATMU registers.

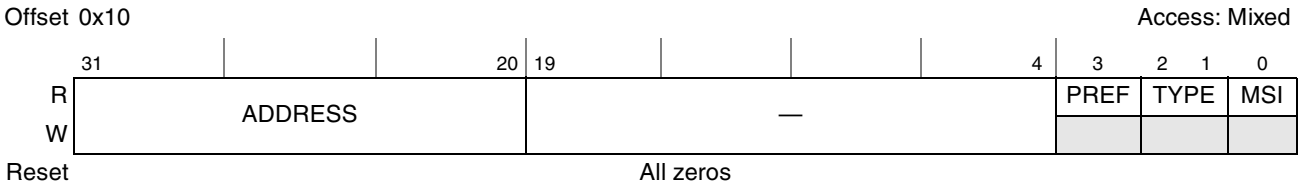


Figure 20-35. PCI Configuration and Status Register Base Address Register (PCSRBAR)

Table 20-34. PCSRBAR Field Descriptions

Bits	Name	Description
31–20	ADDRESS	Indicates the base address that the inbound configuration/run-time window resides at. This window is fixed at 1 Mbyte.
19–4	—	Reserved
3	PREF	Prefetchable
2–1	TYPE	Type. 00 Locate anywhere in 32-bit address space.
0	MSI	Memory space indicator

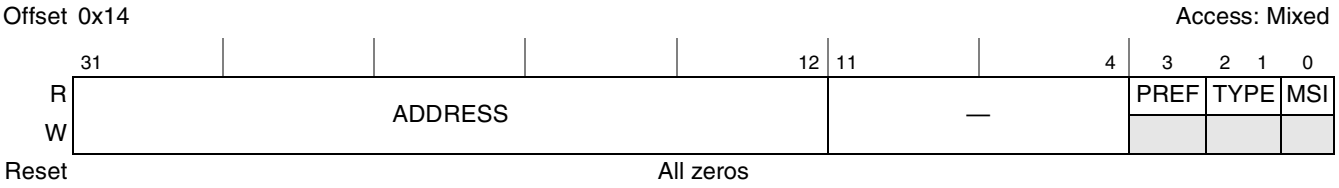


Figure 20-36. 32-Bit Memory Base Address Register

Table 20-35. 32-Bit Memory Base Address Register Field Descriptions

Bits	Name	Description
31–12	ADDRESS	Indicates the base address that the inbound memory window resides at. The number of upper bits that the device allows to be writable is selected through the inbound translation windows.
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable
2–1	TYPE	Type. 00 Locate anywhere in 32-bit address space.
0	MSI	Memory space indicator.

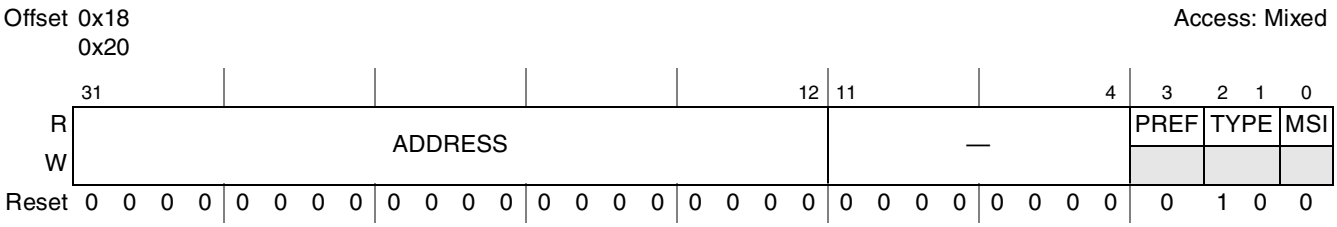


Figure 20-37. 64-Bit Low Memory Base Address Register

Table 20-36. 64-Bit Low Memory Base Address Register Field Descriptions

Bits	Name	Description
31–12	ADDRESS	Indicates the base address that the inbound memory window resides at. The number of upper bits that the device allows to be writable is selected through the inbound translation windows.
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable
2–1	TYPE	Type. 10 Locate anywhere in 64-bit address space.
0	MSI	Memory space indicator

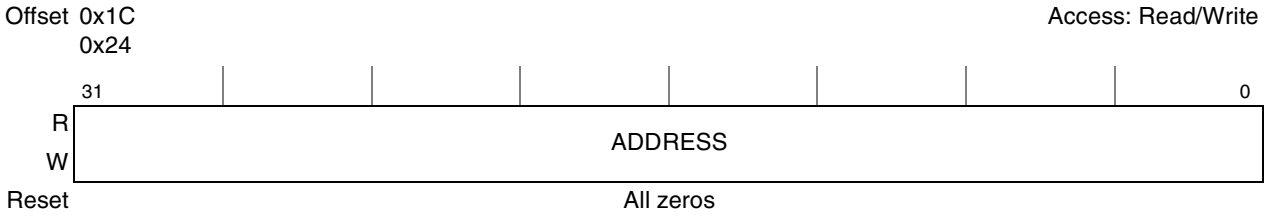


Figure 20-38. 64-Bit High Memory Base Address Register

Table 20-37. Bit Setting for 64-Bit High Memory Base Address Register

Bits	Name	Description
31–0	ADDRESS	Indicates the base address that the inbound memory window resides at. The number of upper bits that the device allows to be writable is selected through the inbound translation windows. If no access to local memory is to be permitted by external masters then all bits are programmed.

20.3.2.12 PCI Subsystem Vendor ID Register

The PCI subsystem vendor ID register is used to identify the subsystem.

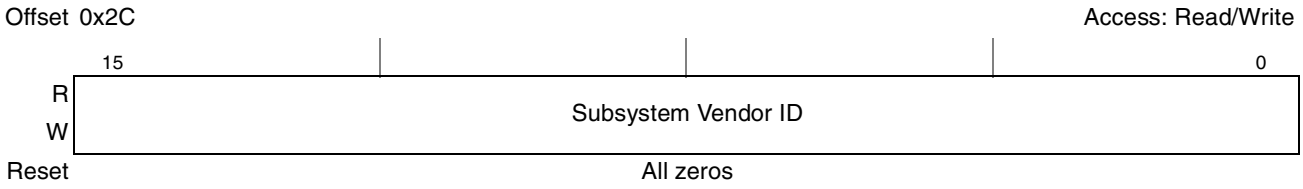


Figure 20-39. PCI Subsystem Vendor ID Register

Table 20-38. PCI Subsystem Vendor ID Register Field Description

Bits	Name	Description
15–0	Subsystem Vendor ID	0x0000

20.3.2.13 PCI Subsystem ID Register

The PCI subsystem ID register is used to identify the subsystem.

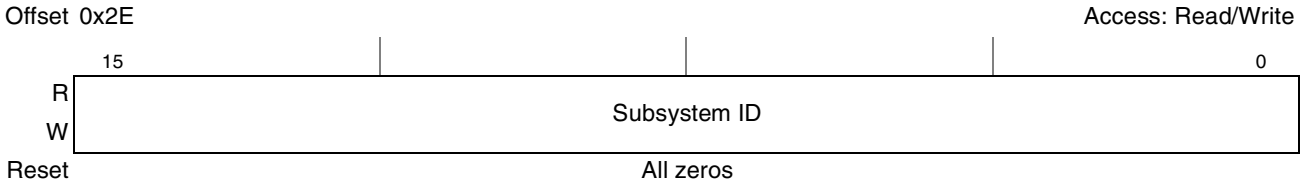


Figure 20-40. PCI Subsystem ID Register

Table 20-39. PCI Subsystem ID Register Field Description

Bits	Name	Description
15–0	Subsystem ID	0x0000

20.3.2.14 PCI Bus Capabilities Pointer Register

The PCI bus capabilities pointer identifies additional functionality supported by the device.

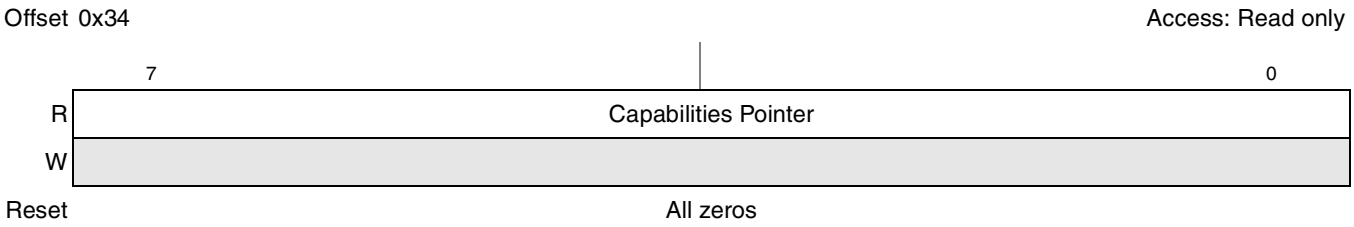


Figure 20-41. PCI Bus Capabilities Pointer Register

Table 20-40. PCI Bus Capabilities Pointer Register Field Description

Bits	Name	Description
7–0	Capabilities Pointer	No additional capabilities

20.3.2.15 PCI Bus Interrupt Line Register

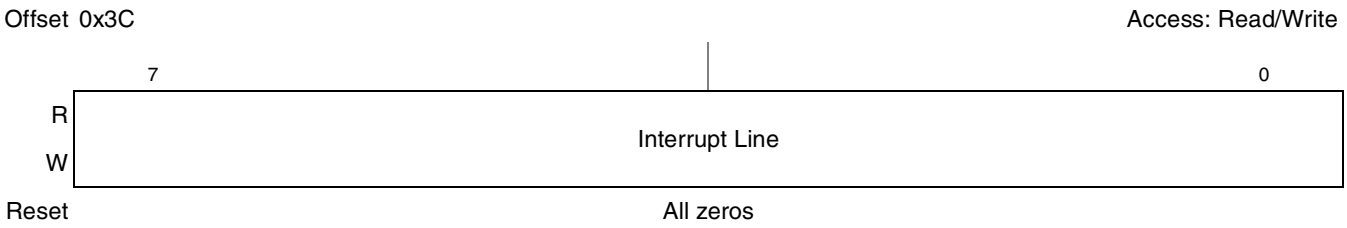


Figure 20-42. PCI Bus Interrupt Line Register

Table 20-41. PCI Bus Interrupt Line Register Field Description

Bits	Name	Description
7–0	Interrupt Line	Used to communicate interrupt line routing information.

20.3.2.16 PCI Bus Interrupt Pin Register

The programmable interrupt controller (PIC) has 12 general purpose interrupt request inputs (IRQ[0:11]) and an interrupt output, $\overline{\text{IRQ\_OUT}}$ (active low, level sensitive), to which all external and most internal interrupt sources (including PCI) can be routed. $\overline{\text{IRQ\_OUT}}$ is mapped to PCI INTA# as a default. Note that this device does not respond to INTACK or special cycle commands on the PCI interfaces.

Offset 0x3D Access: Read only

	7						0
R	Interrupt Pin						
W							
Reset	0	0	0	0	0	0	0	1

Figure 20-43. PCI Bus Interrupt Pin Register

Table 20-42. PCI Bus Interrupt Pin Register Field Description

Bits	Name	Description
7–0	Interrupt pin	PCI_INTA pin selected

20.3.2.17 PCI Bus Minimum Grant Register (MIN_GNT)

Offset 0x3E Access: Read only

	7	0
R	MINGNT	
W		
Reset	All zeros	

Figure 20-44. PCI Bus Minimum Grant Register (MIN_GNT)

Table 20-43. PCI Bus Minimum Grant Register Field Description

Bits	Name	Description
7–0	MINGNT	Specifies the length of the device's burst period (0x00 indicates that this PCI controller has no major requirements for the settings of latency timers.)

20.3.2.18 PCI Bus Maximum Latency Register (MAX_LAT)

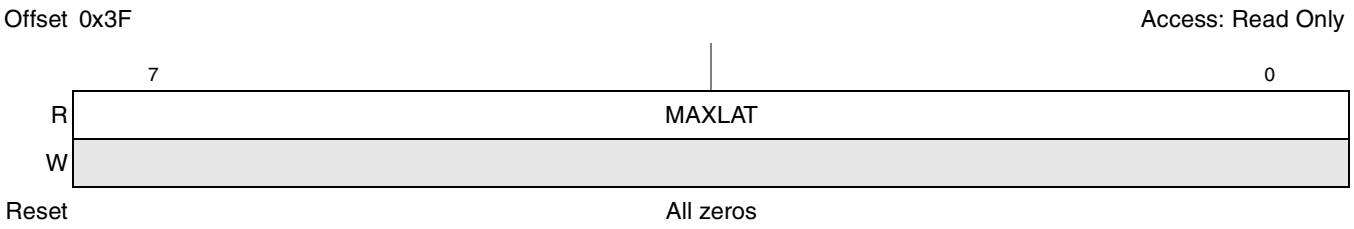


Figure 20-45. PCI Bus Maximum Latency Register (MAX_LAT)

Table 20-44. PCI Bus Maximum Latency Register Field Description

Bits	Name	Description
7–0	MAXLAT	Specifies how often the device needs to gain access to the PCI bus (0x00 indicates that this PCI controller has no major requirements for the settings of latency timers.)

20.3.2.19 PCI Bus Function Register (PBFR)

The 2-byte PCI bus function register is used to determine how different features of the PCI interface in bus 0 are configured. This register is in PCI configuration space at offset 0x44.

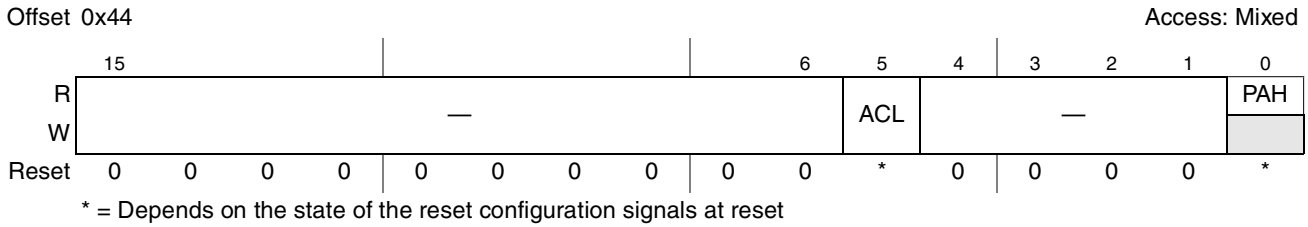


Figure 20-46. PCI Bus Function Register

Table 20-45. PCI Bus Function Register Field Descriptions

Bits	Name	Description
15–6	—	Reserved
5	ACL	Agent configuration lock. Indicates to an external host whether the local processor is doing internal configuration and must be explicitly set and cleared by the local processor during this time. ACL is set during reset if the cfg_cpu_boot configuration input selects the CPU as the configuration owner. (See Section 4.4.3.6, “CPU Boot Configuration.”) This bit is only meaningful in agent mode. 0 PCI interface allows incoming PCI configuration cycles. 1 PCI interface retries all incoming PCI configuration cycles.
4–1	—	Reserved
0	PAH	PCI agent/host. Read-only. Indicates the reset value of the cfg_host_agt configuration signal. 0 PCI interface is in host mode 1 PCI interface is in agent mode

20.3.2.20 PCI Bus Arbiter Configuration Register (PBACR)

The PCI bus arbiter configuration register is used to determine the configuration of the PCI bus arbiter.

Offset 0x46 Access: Mixed

	15	14	13	12	11	7	6	2	1	0
R	PAD	PM	—	PBMD	—		PBMP		—	DP
W										
Reset	All zeros									

Figure 20-47. PCI Bus Arbiter Configuration Register

Table 20-46. PCI Bus Arbiter Configuration Register Field Descriptions

Bits	Name	Description
15	PAD	PCI arbiter disable. Determines if the device is the PCI arbiter on the PCI bus or not. The reset state is determined by the inverse of the cfg_pci*n*_arb configuration input signal when reset is released. 0 Device is the PCI arbiter. 1 Device is not the PCI arbiter. Device presents its request on $\overline{\text{PCI\_REQ0}}$ to the external arbiter and receives its grant on $\overline{\text{PCI\_GNT0}}$.
14	PM	Parking mode. controls which device receives the bus grant when there are no outstanding bus requests and the bus is idle. 0 The bus is parked on the last device to use the bus. 1 The bus is parked on the device.
13	—	Reserved
12	PBMD	PCI broken master disable. Determines if the device ignores the bus requests of an initiator that requests the bus for an excessive period without using the bus. 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 else its request is subsequently ignored. 1 No requests are ignored.
11–7	—	Reserved
6–2	PBMP	PCI bus master priorities. Determines arbitration priority given to different masters on the PCI bus. Bit 6 corresponds to the priority of the master sourcing $\overline{\text{PCI\_REQ0}}$; bit 2 corresponds to the priority of the master sourcing $\overline{\text{PCI\_REQ4}}$. 0 Master *n* is low priority. 1 Master *n* is high priority.
1	—	Reserved
0	DP	Device priority. Determines this device's arbitration priority. 0 Device is low priority. 1 Device is high priority.

20.4 Functional Description

This section describes the functionality of the PCI interface.

20.4.1 PCI Bus Arbitration

PCI bus arbitration is access-based. Bus masters must arbitrate for each access performed on the bus. The PCI bus uses a central arbitration scheme where each master has its own unique request ($\overline{\text{REQ}}$) output and grant ($\overline{\text{GNT}}$) 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 due to arbitration (except when the bus is idle).

The PCI controller provides bus arbitration logic for its master interface and up to five other external PCI bus masters. The on-chip PCI arbiter is independent of host or agent mode. The on-chip PCI arbiter functions in both host and agent modes, or it can be disabled to allow for an external PCI arbiter.

A configuration signal sampled at the negation of the reset signal ($\overline{\text{HRESET}}$) determines if the on-chip PCI arbiter is enabled (high) or disabled (low). The on-chip PCI arbiter can also be enabled or disabled by programming bit 15 of the PCI bus arbitration control register (PBACR[PAD]). Note that the sense of PBACR[PAD] corresponds to the inverse of the configuration signal (that is, when PAD = 0 the arbiter is enabled, and when PA = 1 the arbiter is disabled). See Chapter 4, “Reset, Clocking, and Initialization,” for more information on the reset configuration signals.

If the on-chip PCI arbiter is enabled, a request-grant pair of signals is provided for each external master ($\overline{\text{PCI\_REQ}}$[0:4] and $\overline{\text{PCI\_GNT}}$[0:4]). In addition, there is an internal request/grant pair for the internal master state machine that governs internal accesses to the PCI interface. If the on-chip PCI arbiter is disabled, the PCI controller uses the $\overline{\text{PCI\_REQ0}}$ signal as an output to issue its request to the external arbiter and uses the $\overline{\text{PCI\_GNT0}}$ signal as an input to receive its grant from the external arbiter.

The following sections describe the operation of the on-chip PCI arbiter that arbitrates between external PCI masters and the internal PCI bus master.

20.4.1.1 PCI Bus Arbiter Operation

The on-chip PCI arbiter uses a programmable two-level, round-robin arbitration algorithm. Each of the five external masters, plus the device itself, can be programmed for two priority levels, high or low, using the appropriate bits in the PBACR. Within each priority group, the PCI bus grant is asserted to the next requesting device in numerical order, with the PCI controller positioned before device 0.

Conceptually, the lowest priority device is the master that is currently using the bus, and the highest priority device is the device that follows the current master in numerical order and group priority. This is considered to be a fair algorithm, since a single device cannot prevent other devices from having access to the bus; it automatically becomes the lowest priority device as soon as it begins to use the bus. If a master is not requesting the bus, then its transaction slot is given to the next requesting device within its priority group.

A grant is awarded to the highest priority requesting device as soon as the current master begins a transaction; however, the granted device must wait until the bus is relinquished by the current master before initiating a transaction.

The grant given to a particular device may be removed and awarded to another higher priority device, whenever the higher priority device asserts its request. If the bus is idle when a device requests the bus, then the arbiter withholds the grant for one clock cycle. The arbiter re-evaluates the priorities of all requesting devices and grants the bus to the highest priority device in the following clock cycle. This allows a turnaround clock when a higher priority device is using address stepping or when the bus is parked.

The low-priority group collectively has one bus transaction request slot in the high-priority group. For N high-priority devices and M low-priority devices, each high-priority device is guaranteed at least 1 of N+1 bus transactions and each low-priority device is guaranteed at least 1 of $(N+1) \times M$ bus transactions, with one low-priority device receiving the grant in 1 of N+1 bus transactions. If all devices are programmed to

the same priority level, or if the low-priority group has only one device, the algorithm defaults to give each device an equal number of bus grants in round-robin sequence.

For the example in Figure 20-48, assume that several devices are requesting the bus. If two masters are in the high-priority group and three are in the low-priority group, each high-priority master is guaranteed at least one out of three transaction slots and each low-priority master is guaranteed one out of nine transaction slots.

In Figure 20-48, the grant sequence (with all devices, except device 4 requesting the bus and device 3 being the current master) is 0, 2, , 0, 2, 1, 0, 2, 3, …, and repeating. If device 2 is not requesting the bus, the grant sequence is 0, , 0, 1, 0, 3, …, and repeating. If device 2 requests the bus when device 0 is conducting a transaction and the has the next grant, the has its grant removed and device 2 is awarded the grant since device 2 is higher priority than the when device 0 has the bus.

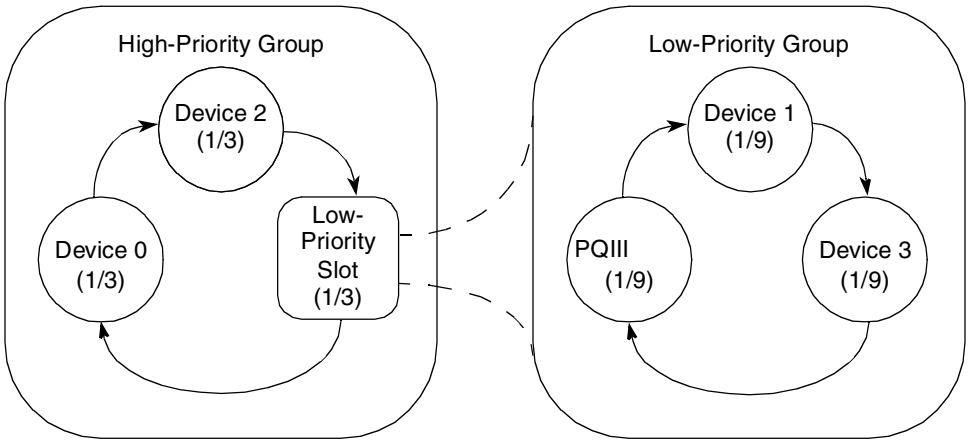


Figure 20-48. PCI Arbitration Example

20.4.1.2 PCI Bus Parking

When no device is using or requesting the bus, the PCI arbiter grants the bus to a selected device. This is known as parking the bus on the selected device. The selected device is required to drive the PCI_AD[31:0], PCI_C/$\overline{\text{BE}}$[0:3], and the PCI parity signals to a stable value, preventing these signals from floating.

The parking mode parameter (PBACR[PM]) determines which device the arbiter selects for parking the PCI bus. If PBACR[PM] = 0 (or if the bus is not idle), then the bus is parked on the last master to use the bus. If the bus is idle and PBACR[PM] = 1, the bus is parked on the PCI controller.

20.4.1.3 Broken Master Lock-Out

The PCI bus arbiter has a feature that allows it to lock out any masters that are broken or ill-behaved. The broken master feature is controlled by programming bit 12 of the PCI bus arbitration control register (0 = enabled, 1 = disabled).

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 REQ signal. Note that disabling the broken master feature is not recommended.

20.4.1.4 Power-Saving Modes and the PCI Arbiter

In the sleep power-saving mode, the clock signal driving SYSCLK can be disabled. If the clock is disabled, the arbitration logic is not able to perform its function. System programmers must park the bus with a device that can sustain the PCI_AD[31:0], PCI_C/BE[3:0], and parity signals prior to disabling the SYSCLK signal. If the bus is parked on the when its clocks are stopped, the sustains the PCI_AD[31:0], PCI_C/BE[3:0], and parity signals in their prior states. In this situation, the only way for another agent to use the PCI bus is by waking the . In nap and doze power-saving modes, the arbiter continues to operate allowing other PCI devices to run transactions.

20.4.2 PCI Bus Protocol

This section provides a general description of the PCI bus protocol. Specific PCI bus transactions are described in Section 20.4.2.7, “PCI Bus Transactions.” Refer to Figure 20-49, Figure 20-50, Figure 20-51, and Figure 20-52 for examples of the transfer-control mechanisms described in this section.

All signals are sampled on the rising edge of the PCI bus clock (SYSCLK). Each signal has a setup and hold aperture with respect to the rising clock edge in which transitions are not allowed. Outside this aperture, signal values or transitions have no significance. See the separate hardware specifications document for specific setup and hold times.

20.4.2.1 Basic Transfer Control

The basic PCI bus transfer mechanism is a burst. A burst is composed of an address phase followed by one or more data phases. Fundamentally, all PCI data transfers are controlled by three signals—PCI_FRAME (frame), PCI_IRDY (initiator ready), and PCI_TRDY (target ready). An initiator asserts PCI_FRAME to indicate the beginning of a PCI bus transaction and negates PCI_FRAME to indicate the end of a PCI bus transaction. An initiator negates PCI_IRDY to force wait cycles. A target negates PCI_TRDY to force wait cycles.

The PCI bus is considered idle when both PCI_FRAME and PCI_IRDY are negated. The first clock cycle in which PCI_FRAME is asserted indicates the beginning of the address phase. The address and bus command code are transferred in that first cycle. The next cycle begins the first of one or more data phases. Data is transferred between initiator and target in each cycle that both PCI_IRDY and PCI_TRDY are asserted. Wait cycles may be inserted in a data phase by the initiator (by negating PCI_IRDY) or by the target (by negating PCI_TRDY).

Once an initiator has asserted PCI_IRDY, it cannot change PCI_IRDY or PCI_FRAME until the current data phase completes regardless of the state of PCI_TRDY. Once a target has asserted PCI_TRDY or PCI_STOP, it cannot change PCI_DEVSEL, PCI_TRDY, or PCI_STOP until the current data phase completes. In simpler terms, once an initiator or target has committed to the data transfer, it cannot change its mind.

When the initiator intends to complete only one more data transfer (which could be immediately after the address phase), PCI_FRAME is negated and PCI_IRDY is asserted (or kept asserted), indicating the initiator is ready. After the target indicates the final data transfer (by asserting PCI_TRDY), the PCI bus may return to the idle state (both PCI_FRAME and PCI_IRDY are negated) unless a fast back-to-back transaction is in progress. In the case of a fast back-to-back transaction, an address phase immediately follows the last data phase.

20.4.2.2 PCI Bus Commands

A PCI bus command is encoded in the PCI_C/BE[3:0] signals during the address phase of a PCI transaction. The bus command indicates to the target the type of transaction the initiator is requesting. Table 20-47 describes the PCI bus commands implemented by the device.

Table 20-47. PCI Bus Commands

PCI_C/BE[3:0]	PCI Bus Command	Supported as an Initiator	Supported as a Target	Definition
0000	Interrupt-acknowledge	Yes	No	A read (implicitly addressing the system interrupt controller). Only one device on the PCI bus should respond to this command; others ignore it. See Section 20.4.2.12.1, “Interrupt-Acknowledge Transactions,” for more information.
0001	Special cycle	Yes	No	Provides a way to broadcast select messages to all devices on the PCI bus. See Section 20.4.2.12.2, “Special-Cycle Transactions,” for more information.
0010	I/O-read	Yes	No	Accesses agents mapped into the PCI I/O space.
0011	I/O-write	Yes	No	Accesses agents mapped into the PCI I/O space.
0100	Reserved[1]	No	No	—
0101	Reserved[1]	No	No	—
0110	Memory-read	Yes	Yes	Accesses either local memory or agents mapped into PCI memory space, depending on the address. When a PCI master issues this command to local memory, the PCI controller (the target) fetches data from the requested address to the end of the cache line (32 bytes) from local memory, even though all of the data may not be requested by (or sent to) the initiator.
0111	Memory-write	Yes	Yes	Accesses either local memory or agents mapped into PCI memory space, depending on the address.
1000	Reserved[1]	No	No	—
1001	Reserved[1]	No	No	—
1010	Configuration-read	Yes	Agent mode only	Accesses the 256-byte configuration space of a PCI agent. A specific agent is selected when its IDSEL signal is asserted during the address phase. See Section 20.4.2.11, “Configuration Cycles,” for details.
1011	Configuration-write	Yes	Agent mode only	
1100	Memory-read-multiple	Yes	Yes	Similar to the memory-read command, but also causes a prefetch of the next cache line (32 bytes).
1101	Dual-address-cycle	Yes	Yes	Used to transfer a 64-bit address (in two 32-bit address cycles) to 64-bit addressable devices.

Table 20-47. PCI Bus Commands (continued)

PCI_C/$\overline{BE}$[3:0]	PCI Bus Command	Supported as an Initiator	Supported as a Target	Definition
1110	Memory-read-line	Yes	Yes	Indicates that an initiator is requesting the transfer of an entire cache line. This occurs only when the processor is performing a burst read. Note that these processors perform burst reads only when the appropriate cache is enabled and the transaction is not cache-inhibited.
1111	Memory-write-and-invalidate	No	Yes	Indicates that an initiator is transferring an entire cache line; if this data is in any cacheable memory, that cache line needs to be invalidated.

[1] Reserved command encodings are reserved for future use. The PCI controller does not respond to these commands.

20.4.2.3 Addressing

PCI defines three physical address spaces—PCI memory space, PCI I/O space, and PCI configuration space. Access to the PCI memory and I/O space is straightforward, although one must take into account the local memory access window and address translation being used. The address translation registers are described in Section 20.3.1, "PCI Memory-Mapped Registers." Access to the PCI configuration space is described in Section 20.4.2.11, "Configuration Cycles."

Address decoding on the PCI bus is performed by every device for every PCI transaction. Each agent is responsible for decoding its own address. PCI supports two types of address decoding—positive decoding and subtractive decoding. For positive decoding, each device looks for accesses in the address range that the device has been assigned. For subtractive decoding, one device on the bus looks for accesses that no other device has claimed. See Section 20.4.2.4, "Device Selection," for information about claiming transactions.

The information contained in the two low-order address bits (PCI_AD[1:0]) varies by the address space (memory, I/O, or configuration). Regardless of the encoding scheme, the two low-order address bits are always included in parity calculations.

20.4.2.3.1 Memory Space Addressing

For memory accesses, PCI defines two types of burst ordering controlled by the two low-order bits of the address—linear incrementing (PCI_AD[1:0] = 0b00) and cache wrap mode (PCI_AD[1:0] = 0b10), as shown in Table 20-48. The other two PCI_AD[1:0] possibilities (0b01 and 0b11) are reserved. As an initiator, the PCI controller always encodes PCI_AD[1:0] = 00 for PCI memory space accesses. As a target, the PCI controller executes a target disconnect after the first data phase completes if PCI_AD[1:0] = 01 or PCI_AD[1:0] = 0b11 during the address phase of a local memory access. See Section 20.4.2.8.2, "Target-Initiated Termination," for more information on target disconnect conditions.

Table 20-48. Supported Combinations of PCI_AD[1:0]

PCI_AD[1:0]		Target		Initiator	
		Read	Write	Read	Write
00	Linear	√	√	√	√
01	Reserved	TD	TD	—	—

Table 20-48. Supported Combinations of PCI_AD[1:0] (continued)

PCI_AD[1:0]		Target		Initiator	
		Read	Write	Read	Write
10	Cache Wrap	√	TD	—	—
11	Reserved	TD	TD	—	—

For linear incrementing mode, the memory address is encoded/decoded using PCI_AD[31:2]. Thereafter, the address is incremented by 4 bytes after each data phase completes until the transaction is terminated or completed (a 4-byte data width per data phase is implied). Note that the two low-order bits on the address bus are included in all parity calculations.

For cache wrap mode (PCI_AD[1:0] = 0b10) reads, the critical memory address is decoded using PCI_AD[31:2]. The address is incremented by 4 bytes after each data phase completes until the end of the cache line is reached. For cache-wrap reads, the address wraps to the beginning of the current cache line and continues incrementing until the entire cache line (32 bytes) is read. The PCI controller does not support cache-wrap write operations and executes a target disconnect after the data phase for the end of the cache line completes for writes with PCI_AD[1:0] = 0b10. That is, the PCI controller does not wrap back to the beginning of the cache line. Note that the two low-order bits on the address bus are included in all parity calculations.

20.4.2.3.2 I/O Space Addressing

For PCI I/O accesses, 32 address signals (PCI_AD[31:0]) are used to provide a byte address. After a target has claimed an I/O access, it must determine 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 of the target, the entire access cannot complete. In this case, the target does not transfer any data and terminates the transaction with a target-abort error. See Section 20.4.2.8.2, “Target-Initiated Termination,” for more information.

20.4.2.3.3 Configuration Space Addressing

PCI supports two types of configuration accesses that use different formats for the PCI_AD[31:0] signals during the address phase. The two low-order bits of the address indicate the format used for the configuration address phase—type 0 (PCI_AD[1:0] = 0b00) or type 1 (PCI_AD[1:0] = 0b01). Both address formats identify a specific device and a specific configuration register for that device. See Section 20.4.2.11, “Configuration Cycles,” for descriptions of the two formats.

20.4.2.4 Device Selection

The $\overline{\text{PCI\_DEVSEL}}$ signal is driven by the target of the current transaction. $\overline{\text{PCI\_DEVSEL}}$ indicates to the other devices on the PCI bus that the target has decoded the address and claimed the transaction. $\overline{\text{PCI\_DEVSEL}}$ may be driven one, two, or three clock cycles (fast, medium, or slow device select timing) following the address phase. Device select timing is encoded into the device’s PCI bus status register. If no agent asserts $\overline{\text{PCI\_DEVSEL}}$ within three clock cycles of $\overline{\text{PCI\_FRAME}}$, the agent responsible for subtractive decoding may claim the transaction by asserting $\overline{\text{PCI\_DEVSEL}}$.

A target must assert PCI_DEVSEL (claim the transaction) before or coincident with any other target response (assert PCI_TRDY, PCI_STOP, or data signals). In all cases except target-abort, once a target asserts PCI_DEVSEL, it must not negate PCI_DEVSEL until PCI_FRAME is negated (with PCI_IRDY asserted) and the last data phase has completed. For normal termination, negation of PCI_DEVSEL coincides with the negation of PCI_TRDY or PCI_STOP.

If the first access maps into a target's address range, that target asserts PCI_DEVSEL to claim the access. However, if the initiator attempts to continue the burst access across the resource boundary, then the target must issue a target disconnect.

The PCI controller is hardwired for fast device select timing (PCI bus status register [10–9] = 0b00). Therefore, when the PCI controller is the target of a transaction (local memory access or configuration register access), it asserts PCI_DEVSEL one clock cycle following the address phase.

As an initiator, if the PCI controller does not detect the assertion of PCI_DEVSEL within four clock cycles after the address phase (that is, five clock cycles after it asserts PCI_FRAME), it terminates the transaction with a master-abort termination; see Section 20.4.2.8.1, "Master-Initiated Termination."

20.4.2.5 Byte Alignment

The byte enable signals of the PCI bus (PCI_C/BE[3:0], during a data phase) are used to determine which byte lanes carry meaningful data. The byte enable signals may enable different bytes for each of the data phases. The byte enables are valid on the edge of the clock that starts each data phase and stay valid for the entire data phase. Note that parity is calculated for all bytes regardless of the state of the byte enable signals. See Section 20.4.2.13.1, "PCI Parity," for more information.

If the PCI controller, as a target, detects no byte enables asserted, it completes the current data phase with no permanent change. This implies that on a read transaction, the PCI controller expects that the data is not changed, and on a write transaction, the data is not stored.

20.4.2.6 Bus Driving and Turnaround

To avoid contention, a turnaround cycle is required on all signals that may be driven by more than one agent. The turnaround cycle occurs at different times for different signals. The PCI_IRDY, PCI_TRDY, PCI_DEVSEL, and PCI_STOP signals use the address phase as their turnaround cycle. PCI_FRAME,PCI_C/BE[3:0], and PCI_AD[31:0] signals use the idle cycle between transactions (when both PCI_FRAME and PCI_IRDY are negated) as their turnaround cycle. PCI_PERR has a turnaround cycle on the fourth clock cycle after the last data phase.

The PCI address/data signals, PCI_AD[31:0], are driven to a stable condition during every address/data phase. Even when the byte enables indicate that byte lanes carry meaningless data, the signals carry stable values. Parity is calculated on all bytes regardless of the byte enables. See Section 20.4.2.13.1, "PCI Parity," for more information.

20.4.2.7 PCI Bus Transactions

This section provides descriptions of the PCI bus transactions. All bus transactions follow the protocol as described in Section 20.4.2, “PCI Bus Protocol.” Read and write transactions are similar for the memory and I/O spaces, so they are described as generic read transactions and generic write transactions.

The timing diagrams in this section show the relationship of significant signals involved in bus transactions. When a signal is drawn as a solid line, it is actively being driven by the current master or target. When a signal is drawn as a dashed line, no agent is actively driving it. High-impedance signals are indicated to have indeterminate values when the dashed line is between the two rails.

The terms ‘edge’ and ‘clock edge’ always refer to the rising edge of the clock. The terms ‘asserted’ and ‘negated’ always refer to the globally visible state of the signal on the clock edge, and not to signal transitions. ‘ ’ represents a turnaround cycle in the timing diagrams.

20.4.2.7.1 PCI Read Transactions

This section describes PCI single-beat read transactions and PCI burst read transactions.

A read transaction starts with the address phase, occurring when an initiator asserts PCI_FRAME. During the address phase, PCI_AD[31:0] contains a valid address and PCI_C/BE[3:0] contains a valid bus command.

The first data phase of a read transaction requires a turnaround cycle. This allows the transition from the initiator driving PCI_AD[31:0] as address signals to the target driving PCI_AD[31:0] as data signals. The turnaround cycle is enforced by the target with the TRDY signal. The target provides valid data at the earliest one cycle after the turnaround cycle. The target must drive the PCI_AD[31:0] signals when PCI_DEVSEL is asserted.

During the data phase, the PCI_C/BE[3:0] signals indicate which byte lanes are involved in the current data phase. A data phase may consist of a data transfer and wait cycles. The PCI_C/BE[3:0] signals remain actively driven for both reads and writes from the first clock of the data phase through the end of the transaction.

A 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 PCI_IRDY or PCI_TRDY is negated, a wait cycle is inserted and no data is transferred. The initiator indicates the last data phase by negating PCI_FRAME when PCI_IRDY is asserted. The transaction is considered complete when data is transferred in the last data phase.

Figure 20-49 illustrates a PCI single-beat read transaction.

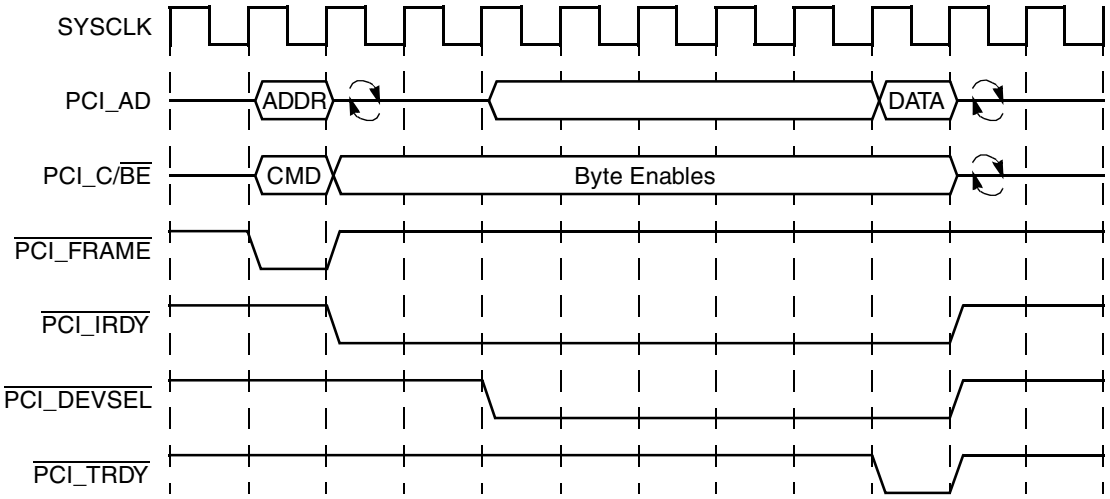


Figure 20-49. PCI Single-Beat Read Transaction

Figure 20-50 illustrates a PCI burst read transaction.

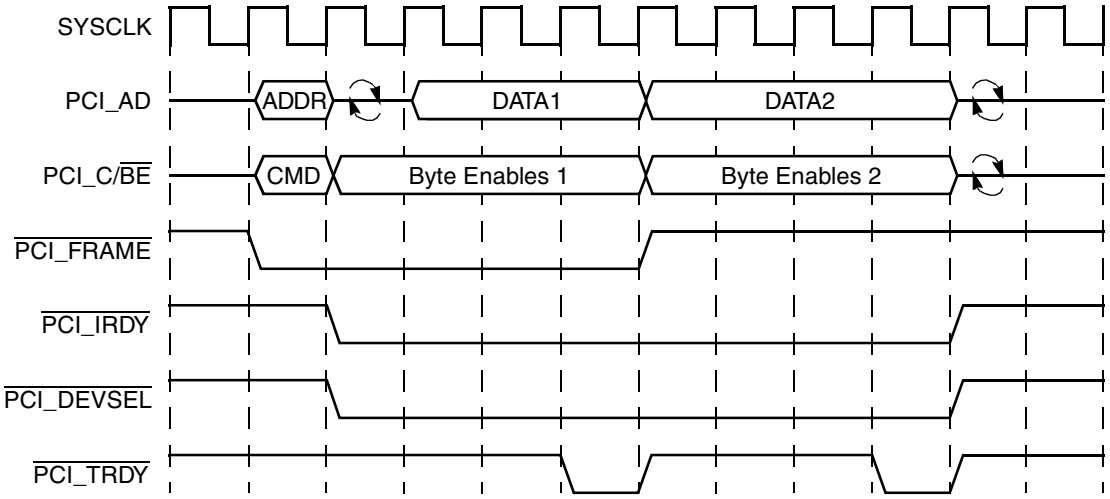


Figure 20-50. PCI Burst Read Transaction

20.4.2.7.2 PCI Write Transactions

This section describes PCI single-beat write transactions, and PCI burst write transactions. A PCI write transaction starts with the address phase, occurring when an initiator asserts PCI_FRAME. 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. The data phases are the same for both read and write transactions. Although not shown in the figures, the initiator must drive the PCI_C/BE[3:0] signals, even if the initiator is not ready to provide valid data (PCI_IRDY negated).

Figure 20-51 illustrates a PCI single-beat write transaction.

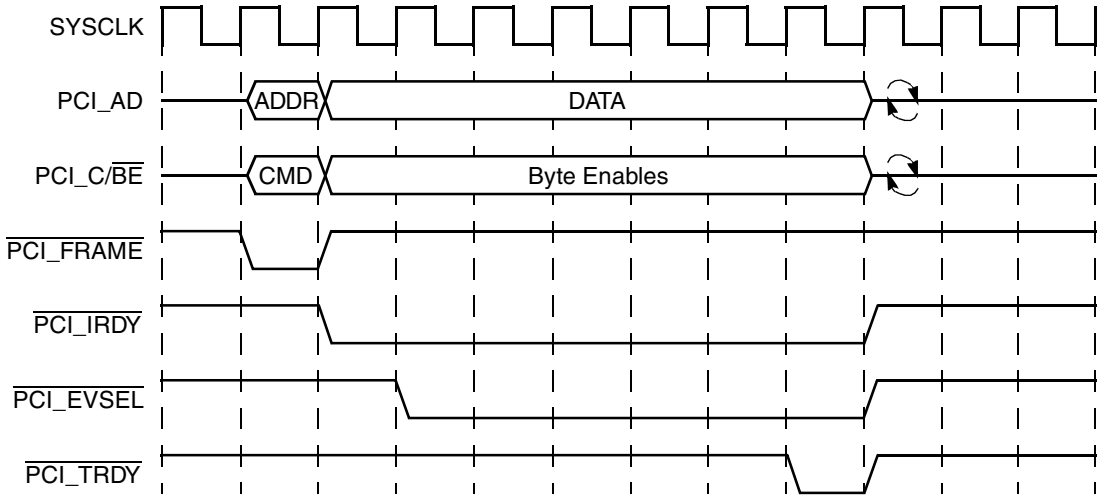


Figure 20-51. PCI Single-Beat Write Transaction

Figure 20-52 illustrates a PCI burst write transaction.

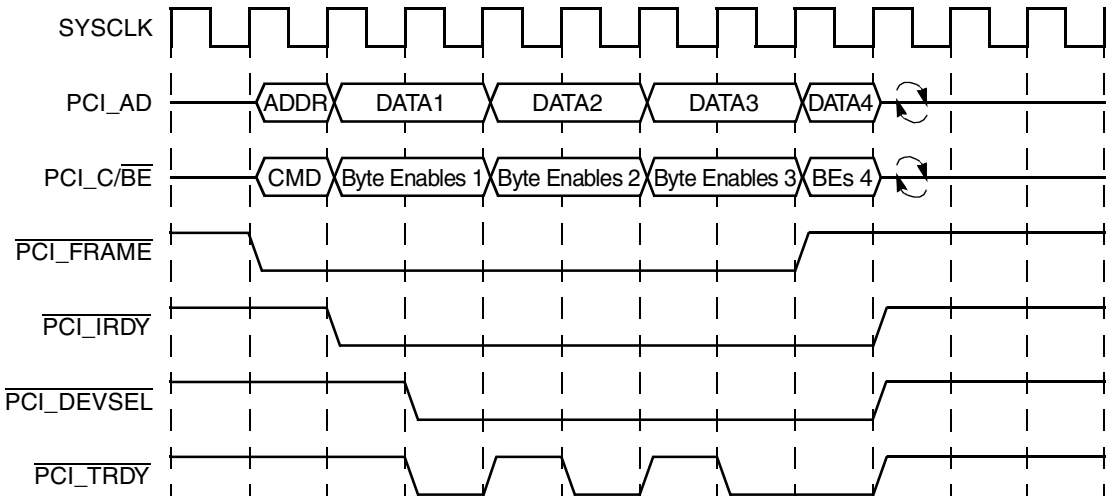


Figure 20-52. PCI Burst Write Transaction

20.4.2.8 Transaction Termination

A PCI transaction may be terminated by either the initiator or the target. The initiator is ultimately responsible for concluding all transactions, regardless of the cause of the termination. All transactions are concluded when PCI_FRAME and PCI_IRDY are both negated, indicating the bus is idle.

20.4.2.8.1 Master-Initiated Termination

Normally, a master initiates termination by negating PCI_FRAME and asserting PCI_IRDY. This indicates to the target that the final data phase is in progress. The final data transfer occurs when both PCI_TRDY and PCI_IRDY are asserted. The transaction is considered complete when data is transferred

in the last data phase. After the final data phase, both PCI_FRAME and PCI_IRDY are negated (the bus becomes idle).

There are three types of master-initiated termination:

- Completion—Refers to termination when the initiator has concluded its intended transaction. This is the most common reason for termination.
- Timeout—Refers to termination when the initiator loses its bus grant (GNT*n* is negated), and its internal latency timer has expired. The intended transaction is not necessarily concluded.
- Master-abort—An abnormal case of master-initiated termination. If no device (including the subtractive decoding agent) asserts PCI_DEVSEL to claim a transaction, the initiator terminates the transaction with a master-abort. For a master-abort termination, the initiator negates PCI_FRAME and then negates PCI_IRDY on the next clock. If a transaction is terminated by master-abort (except for a special-cycle command), the received master-abort bit (bit 13) of the PCI bus status register is set.

As an initiator, if the PCI controller does not detect the assertion of PCI_DEVSEL within four clock cycles following the address phase (five clock cycles after asserting PCI_FRAME), it terminates the transaction with a master-abort.

20.4.2.8.2 Target-Initiated Termination

By asserting the PCI_STOP signal, a target may request that the initiator terminate the current transaction. Once asserted, the target holds PCI_STOP asserted until the initiator negates PCI_FRAME. 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. However, if PCI_TRDY is negated when PCI_STOP is asserted, it indicates that the target does not transfer any more data; therefore, the initiator does not wait for a final data transfer as it would in a completion termination.

When a transaction is terminated by PCI_STOP, the initiator must negate its REQ*n* signal for a minimum of two PCI clock cycles, (one corresponding to when the bus goes to the idle state (PCI_FRAME and PCI_IRDY negated)). If the initiator intends to complete the transaction, it can reassert its REQ*n* immediately following the two clock cycles. If the initiator does not intend to complete the transaction, it can assert REQ*n* whenever it needs to use the PCI bus again.

There are three types of target-initiated termination:

- Disconnect—Disconnect refers to termination requested because the target is temporarily unable to continue bursting. Disconnect implies that some data has been transferred. The initiator may restart the transaction at a later time starting with the address of the next untransferred data. (That is, data transfer may resume where it left off.)
- Retry—Retry refers to termination requested because the target is currently in a state where it is unable to process the transaction. Retry implies that no data was transferred. The initiator may start the entire transaction over again at a later time. Note that the *PCI Local Bus Specification*, Rev. 2.2 requires that all retried transactions must be completed.
- Target-Abort—Target-abort is an abnormal case of target-initiated termination. Target-abort is used when a fatal error has occurred or when a target can never respond.

As a target, the PCI controller terminates a transaction with a target disconnect due to the following:

- It is unable to respond within eight PCI clock cycles (not including the first data phase).
- The transaction is attempting to cross a 4-Kbyte boundary.
- A single beat of data has been transferred and the inbound ATMU is marked non-prefetchable.
- The end of a cache line has been transferred for a cache-wrap mode write transaction. See Section 20.4.2.3.1, “Memory Space Addressing,” for more information.

As a target, the PCI controller responds to a transaction with a retry due to the following:

- The 32-clock latency timer has expired, and the first data phase has not begun.
- There is no more internal buffer space available for an inbound transaction.

Target-abort is indicated by asserting $\overline{\text{PCI\_STOP}}$ and negating $\overline{\text{PCI\_DEVSEL}}$. This indicates that the target requires termination of the transaction and does not want the transaction retried. If a transaction is terminated by target-abort, the received target-abort bit (bit 12) of the initiator’s bus status register and the signaled target-abort bit (bit 11) of the target’s bus status register are set. Note that any data transferred in a target-aborted transaction may be corrupt.

For PCI writes to local memory, if an address parity error or data parity error occurs, the PCI controller aborts the transaction internally, but continues the transaction on the PCI bus.

Figure 20-53 shows several target-initiated terminations.

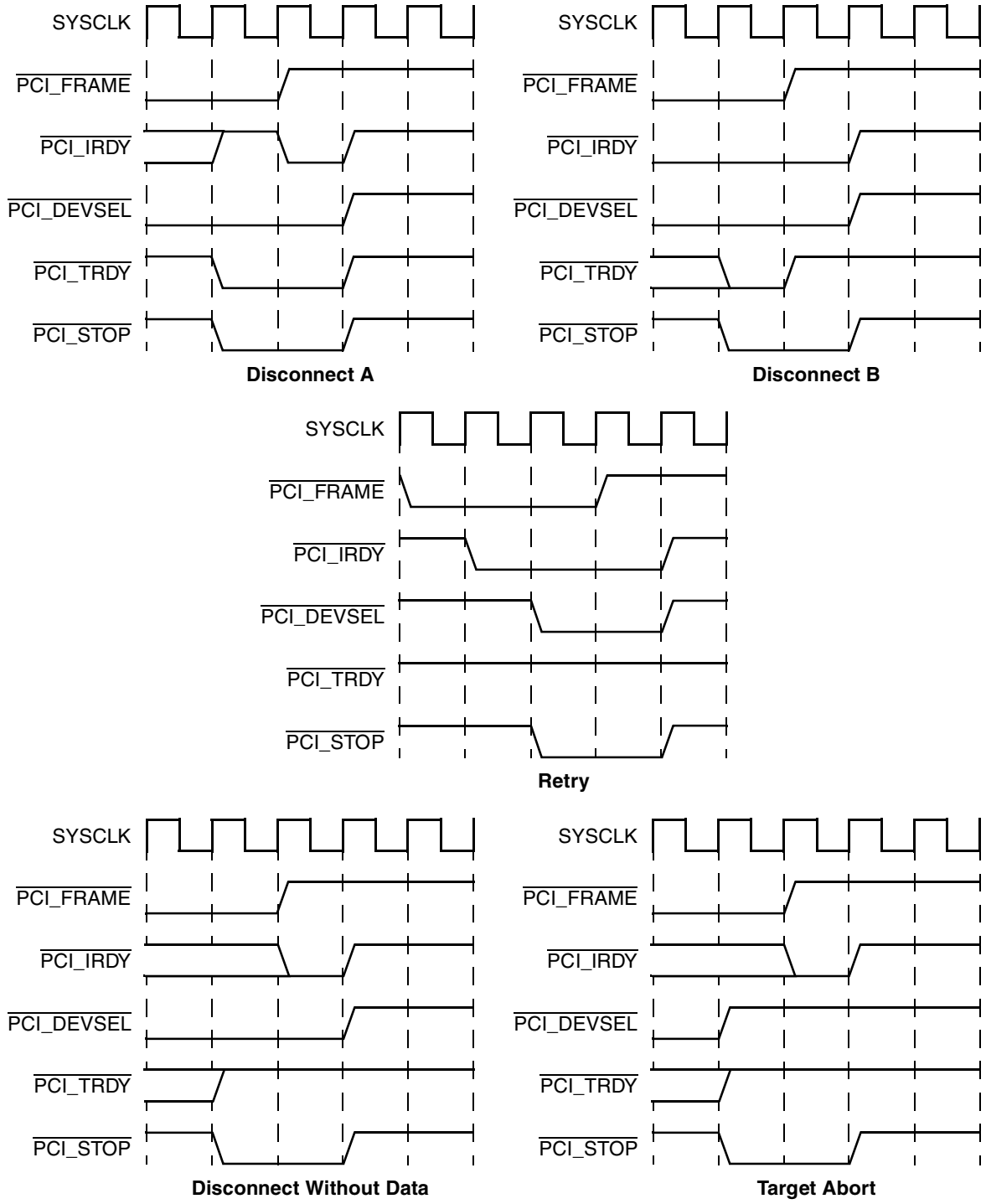


Figure 20-53. PCI Target-Initiated Terminations

The three disconnect terminations are unique in the data transferred at the end of the transaction. For disconnect A, the initiator is negating PCI_IRDY when the target asserts PCI_STOP and data is transferred only at the end of the current data phase. For disconnect B, the target negates PCI_TRDY one clock after

it asserts PCI_STOP, indicating that the target can accept the current data, but no more data can be transferred. For disconnect-without-data, the target asserts PCI_STOP when PCI_TRDY is negated indicating that the target cannot accept any more data.

20.4.2.9 Fast Back-to-Back Transactions

The PCI bus allows fast back-to-back transactions by the same master. During a fast back-to-back transaction, the initiator starts the next transaction immediately without an idle state. The last data phase completes when PCI_FRAME is negated, and PCI_IRDY and PCI_TRDY are asserted. The current master starts another transaction in the clock cycle immediately following the last data transfer for the previous transaction.

Fast back-to-back transactions must avoid contention on the PCI_TRDY, PCI_DEVSEL, PCI_PERR, and PCI_STOP signals. There are two types of fast back-to-back transactions—those that access the same target and those that access multiple targets sequentially. The first type places the burden of avoiding contention on the initiator; the second type places the burden of avoiding contention on all potential targets.

As an initiator, the PCI controller does not perform any fast back-to-back transactions. As a target, the PCI controller supports both types of fast back-to-back transactions.

During fast back-to-back transactions, the PCI controller monitors the bus states to determine if it is the target of a transaction. If the previous transaction was not directed to the PCI controller but the current transaction is directed at the PCI controller, it delays the assertion of PCI_DEVSEL (as well as PCI_TRDY, PCI_STOP, and PCI_PERR) for one clock cycle to allow the other target to stop driving the bus.

20.4.2.10 Dual Address Cycles

The PCI controller supports dual address cycle (DAC) commands (64-bit addressing on PCI bus) as both an initiator and a target. 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 block supports single-beat and burst DAC transactions.

For the case of the local processor, DAC generation depends on the setting of the POTEARx. If the POTEARx are programmed with nonzero values and a transaction from the local processor core hits in one of the outbound windows, a DAC transaction is generated on the PCI bus with the translated lower 32-bit addresses. Refer to Section 20.3.1.2, “PCI ATMU Outbound Registers,” for more information.

The timing sequence of the PCI signals for single-beat DAC reads is shown in Figure 20-54.

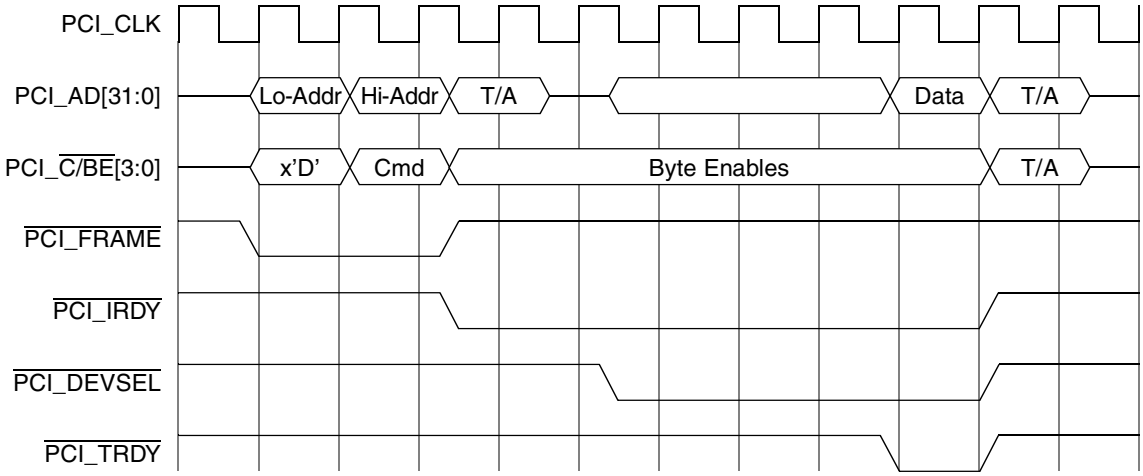


Figure 20-54. DAC Single-Beat Read Example

The timing for a DAC burst read is shown in Figure 20-55.

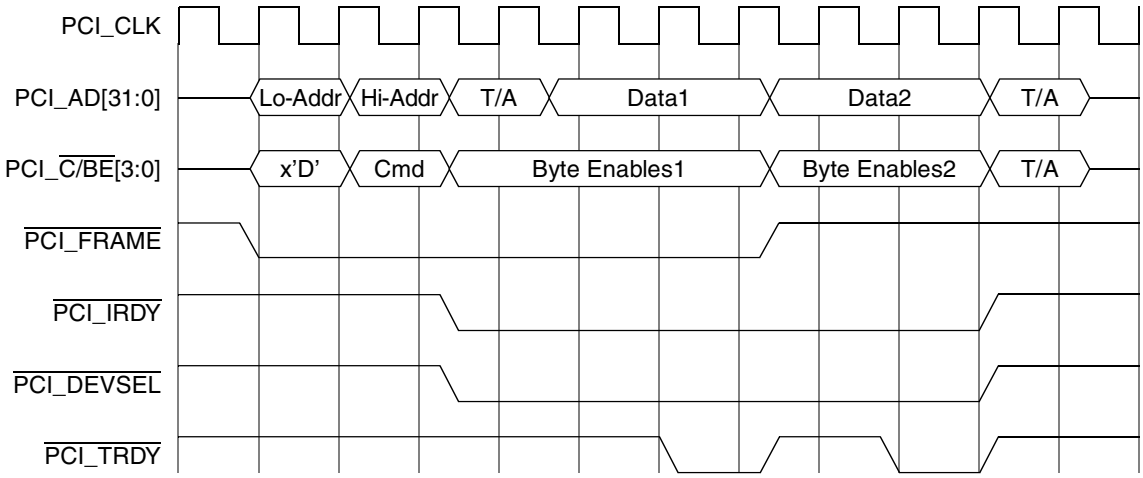


Figure 20-55. DAC Burst Read Example

Figure 20-56 and Figure 20-57 show timing examples for single-beat DAC writes and burst DAC writes, respectively.

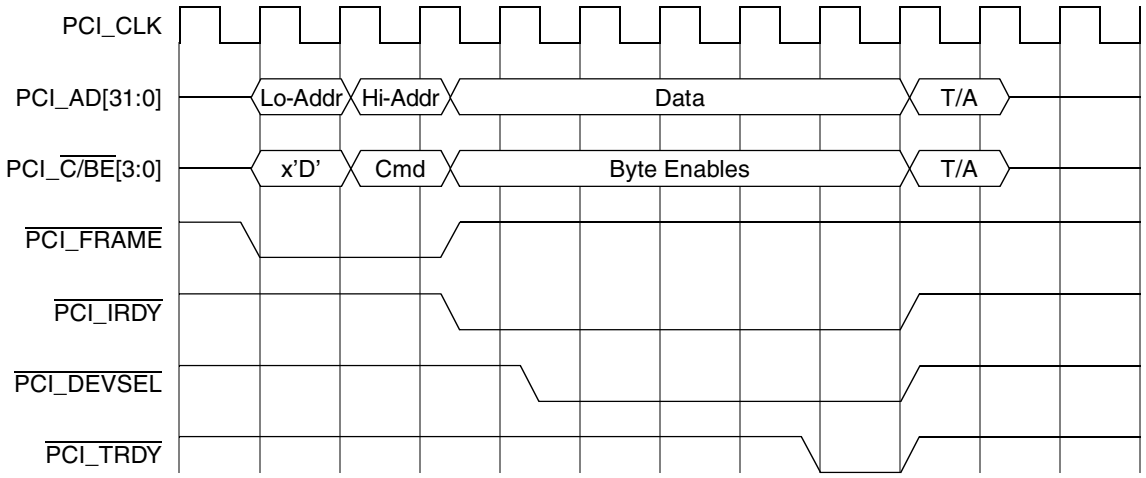


Figure 20-56. DAC Single-Beat Write Example

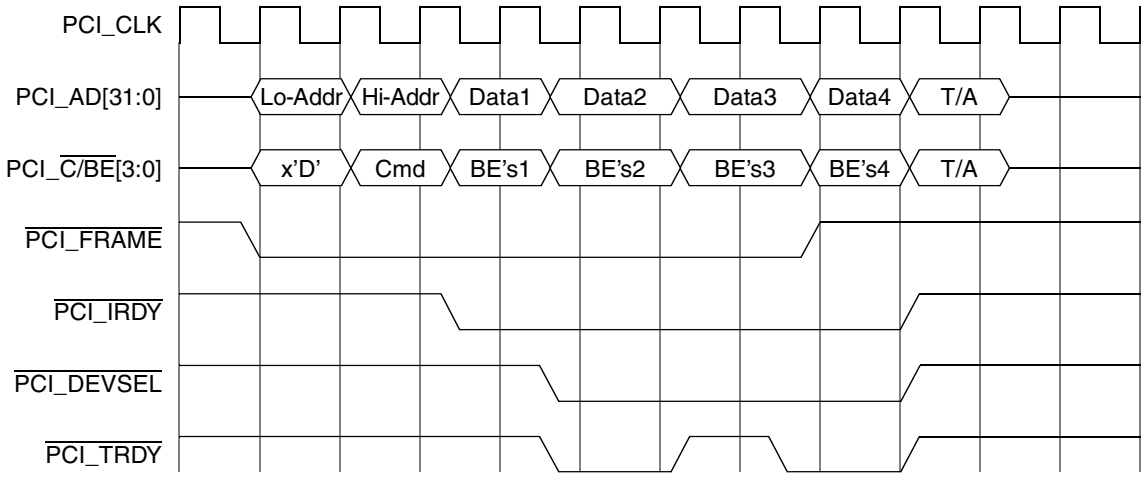


Figure 20-57. DAC Burst Write Example

20.4.2.11 Configuration Cycles

This section describes PCI configuration cycles used for configuring standard PCI devices. The PCI configuration space of any device is intended for configuration, initialization, and catastrophic error-handling functions only. Access to the PCI configuration space should be limited to initialization and error-handling software.

20.4.2.11.1 PCI Configuration Space Header

The first 64 bytes of the 256-byte configuration space consists of a predefined header that every PCI device must support. The predefined header for all PCI devices is shown in Figure 20-58. The first 16 bytes of the predefined header are defined the same for all PCI devices; the remaining 48 bytes of the header may have differing layouts depending on the function of the device. Most PCI devices use the configuration header

layout shown in Figure 20-58. The rest of the 256-byte configuration space is device-specific. The PCI header specific to the PCI controller is described in Section 20.3.2, “PCI Configuration Header.”

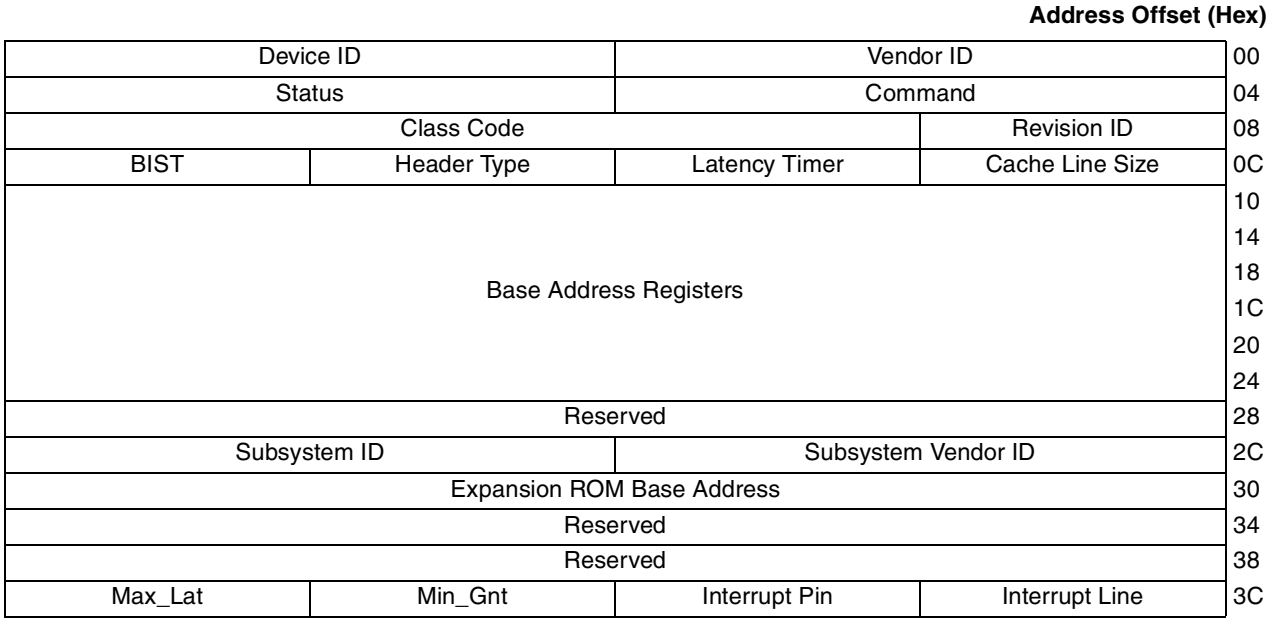


Figure 20-58. Standard PCI Configuration Header

Table 20-49 summarizes the configuration header registers. Detailed descriptions of these registers are provided in the *PCI Local Bus Specification*, Rev. 2.2.

Table 20-49. PCI Configuration Space Header Summary

Address Offset (Hex)	Register Name	Description
0x00	Vendor ID	Identifies the manufacturer of the device (assigned by the PCI SIG (special-interest group) to ensure uniqueness).
0x02	Device ID	Identifies the particular device (assigned by the vendor).
0x04	Command	Provides coarse control over a device’s ability to generate and respond to PCI bus cycles
0x06	Status	Records status information for PCI bus-related events
0x08	Revision ID	Specifies a device-specific revision code (assigned by vendor)
0x09	Class code	Identifies the generic function of the device and (in some cases) a specific register-level programming interface
0x0C	Cache line size	Specifies the system cache line size in 32-bit units
0x0D	Latency timer	Specifies the value of the latency timer in PCI bus clock units for the device when acting as an initiator
0x0E	Header type	Bits 0–6 identify the layout of bytes 0x10–0x3F; bit 7 indicates a multifunction device. The most common header type (0x00) is shown in Figure 20-58 and in this table.
0x0F	BIST	Optional register for control and status of built-in self test (BIST)
0x10–0x27	Base address registers	Address mapping information for memory and I/O space

Table 20-49. PCI Configuration Space Header Summary (continued)

Address Offset (Hex)	Register Name	Description
0x28	—	Reserved for future use
0x2C	Subsystem Vendor ID	Identifies the subsystem vendor ID
0x2E	Subsystem ID	Identifies the subsystem ID
0x30	Expansion ROM base address	Base address and size information for expansion ROM contained in an add-on board
0x34, 0x38	—	Reserved for future use
0x3C	Interrupt line	Contains interrupt line routing information
0x3D	Interrupt pin	Indicates which interrupt pin the device (or function) uses
0x3E	Min_Gnt	Specifies the length of the device's burst period in 0.25 µs units
0x3F	Max_Lat	Specifies how often the device needs access to the bus in 0.25 µs units

To access the configuration space, a 32-bit value must be written to the PCI CFG_ADDR register that specifies the target PCI bus, the target device on that bus, and the configuration register to be accessed within that device. A read or write to the PCI CFG_DATA register causes the host bridge to translate the access into a PCI configuration cycle (provided the enable bit in CONFIG_ADDR is set and the device number is not 0b1_1111).

See Section 20.3.1.1.1, "PCI Configuration Address Register (CFG_ADDR)," for details on PCI CFG_ADDR and Section 20.3.1.1.2, "PCI Configuration Data Register (CFG_DATA)," for details on PCI CFG_DATA.

20.4.2.11.2 Host Accessing the PCI Configuration Space

Power Architecture processor accesses to the PCI CFG_DATA register should use the load/store with byte-reversed instructions.

Example: Configuration sequence, 4-byte data read from the revision ID/standard programming interface/subclass code/class code registers at address offset 0x08 of the PCI configuration header (device 0 on the PCI bus 0 is the PCI controller itself).

```
Initial values:
         r0 contains 0x8000_0008
         r1 contains CCSRBAR + BlockBase + 0x000 (Address of PCI CFG_ADDR register)
         r2 contains CCSRBAR + BlockBase + 0x004 (Address of PCI CFG_DATA register)
         r3 contains 0xFFFF_FFFF
         Register at 0x08 contains 0x9988_7766 (0x0B to 0x08)
Code sequence:
         stw r0, 0 (r1)
         ld r3, 0 (r2)
Results:
         Address CCSRBAR + BlockBase + 0x000 contains 0x8000_0008
         Register r3 contains 0x6677_8899
```

20.4.2.11.3 Agent Accessing the PCI Configuration Space

When this device is configured as an agent device, it responds to a remote host-generated PCI configuration cycle. This is indicated by decoding the configuration command along with PCI's IDSEL being asserted. When the PCI controller detects an access to PCI CFG_DATA, it checks the enable flag and the device number in the PCI CFG_ADDR register. If the enable bit is set, and the device number is not 0b1_1111, the PCI controller performs a configuration cycle translation function and runs a configuration-read or configuration-write transaction on the PCI bus. The device number 0b1_1111 is used for performing interrupt-acknowledge and special-cycle transactions. See Section 20.4.2.12, “Other Bus Transactions,” for more information. If the bus number corresponds to the local PCI bus (bus number = 0x00), the PCI controller performs a type 0 configuration cycle translation. If the bus number indicates a remote PCI bus (that is, nonlocal), the PCI controller performs a type 1 configuration cycle translation.

Note that in the following examples, the data in the configuration register is shown in little-endian order. This is because all the PCI registers are intrinsically little-endian. External PCI masters that use the local address map to access configuration space do not need to reverse bytes since byte lane redirection from the little-endian PCI bus is performed internally.

Example: Configuration sequence, 4-byte data write to PCI register at address offset 0x14 of Device 1 on PCI bus 0.

```
Initial values:
         r0 contains 0x8000_0814
         r1 contains CCSRBAR + BlockBase + 0x000 (Address of PCI CFG_ADDR register)
         r2 contains CCSRBAR + BlockBase + 0x004 (Address of PCI CFG_DATA register)
         r3 contains 0x1122_3344
         Register at 0x14 contains 0xFFFF_FFFF (0x17 to 0x14)
Code sequence:
         stw r0, 0 (r1) // Update PCI CFG_ADDR register to point to
                        //register offset 0x14 of device 1.
         stwbrx r3, 0 (r2)
Results:
         Address CCSRBAR + BlockBase + 0x000 contains 0x8000_0814
         Register at 0x14 contains 0x1122_3344 (0x17 to 0x14)
```

Example: Configuration sequence, 2-byte data write to PCI register at address offset 0x1C of Device 1 on PCI bus 0.

```
Initial values:
         r0 contains 0x8000_081C
         r1 contains CCSRBAR + BlockBase + 0x000
         r2 contains CCSRBAR + BlockBase + 0x004
         r3 contains 0xDDCC_BBAA
         Register at 0x1C contains 0xFFFF_FFFF (0x1F to 0x1C)
Code sequence:
         stw r0, 0 (r1)
         sthbrx r3, 0 (r2)
Results:
         Address CCSRBAR + BlockBase + 0x000 contains 0x8000_081C
         Register at 0x1C contains 0xFFFF_BBAA (0x1F to 0x1C)
```

20.4.2.11.4 PCI Type 0 Configuration Translation

Figure 20-59 shows the PCI type 0 translation function performed on the contents of the PCI CFG_ADDR register to the PCI_AD[31:0] signals on the PCI bus during the address phase of the configuration cycle.

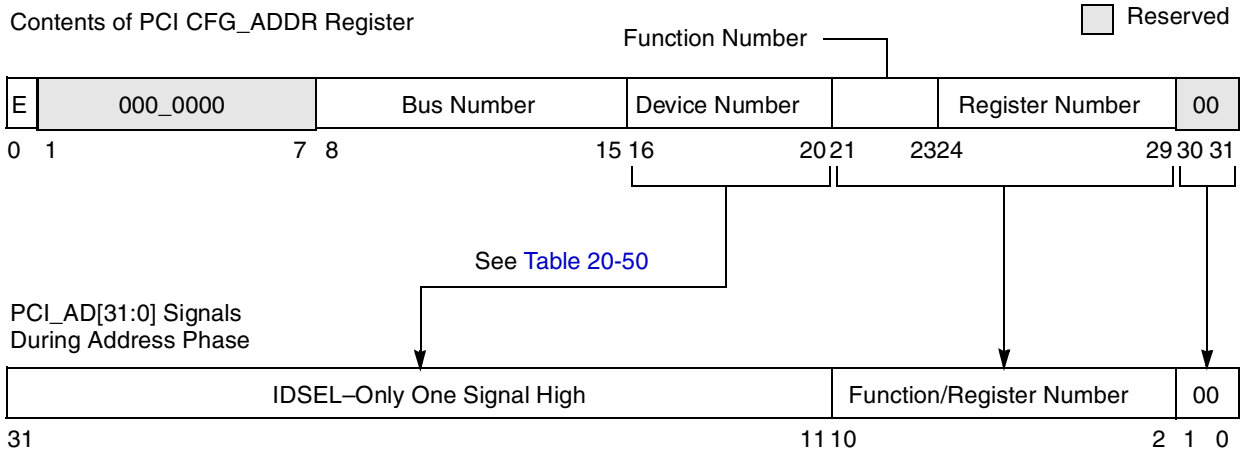


Figure 20-59. PCI Type 0 Configuration Translation

For PCI type 0 configuration cycles, the PCI controller translates the device number field of the PCI CFG_ADDR register into a unique IDSEL signal for up to 21 different devices. Each device connects its IDSEL input to one of the PCI_AD[31:11] signals. For PCI type 0 configuration cycles, the PCI controller translates the device number to AD*n* as shown in Table 20-50.

Table 20-50. PCI Type 0 Configuration—Device Number to AD*n* Translation

Device Number		AD*n* Used for IDSEL	Device Number		AD*n* Used for IDSEL
Binary	Decimal		Binary	Decimal	
0b0_0000	0	—[1]	0b1_0100	20	AD20
0b0_0001–0b0_1001	1–9	—[2]	0b1_0101	21	AD21
0b0_1010	10	AD31	0b1_0110	22	AD22
0b0_1011	11	AD11	0b1_0111	23	AD23
0b0_1100	12	AD12	0b1_1000	24	AD24
0b0_1101	13	AD13	0b1_1001	25	AD25
0b0_1110	14	AD14	0b1_1010	26	AD26
0b0_1111	15	AD15	0b1_1011	27	AD27
0b1_0000	16	AD16	0b1_1100	28	AD28
0b1_0001	17	AD17	0b1_1101	29	AD29
0b1_0010	18	AD18	0b1_1110	30	AD30
0b1_0011	19	AD19	0b1_1111[3]	31	—

[1] No external configuration transaction takes place; rather, internal registers are accessed.
[2] No IDSEL line asserted. Type0 configuration transaction is run, but ends with a master abort since no device responds.
[3] A device number of all ones indicates a PCI special-cycle or interrupt-acknowledge transaction.

For PCI type 0 translations, the function number and register number fields are copied without modification onto the PCI_AD[10:2] signals during the address phase. The PCI_AD[1:0] signals are

driven to 0b00 during the address phase for type 0 configuration cycles. The PCI controller implements address stepping on configuration cycles so that the target's IDSEL, which is connected directly to one of the PCI_AD lines, reaches a stable value. This means that a valid address and command are driven on PCI_AD[31:0] and PCI_C/$\overline{\text{BE}}$[3:0] one clock cycle before the assertion of $\overline{\text{PCI\_FRAME}}$.

20.4.2.11.5 Type 1 Configuration Translation

For type 1 translations, the PCI controller copies the 30 high-order bits of the PCI CFG_ADDR register (without modification) onto the PCI_AD[31:2] signals during the address phase. The PCI controller automatically translates PCI_AD[1:0] into 0b01 during the address phase to indicate a type 1 configuration cycle.

20.4.2.12 Other Bus Transactions

There are two other PCI transactions that the PCI controller supports—interrupt acknowledge and special cycles. As an initiator, the PCI controller may initiate both interrupt acknowledge and special-cycle transactions; however, as a target, the PCI controller ignores interrupt-acknowledge and special-cycle transactions. Both transactions make use of the PCI CFG_ADDR and PCI CFG_DATA registers described in Section 20.4.2.11.3, "Agent Accessing the PCI Configuration Space."

20.4.2.12.1 Interrupt-Acknowledge Transactions

The PCI bus supports an interrupt-acknowledge transaction. The interrupt-acknowledge command is a read operation implicitly addressed to the system interrupt controller. Note that the PCI interrupt-acknowledge command does not address the device's PIC processor interrupt-acknowledge register and does not return the interrupt vector address from the PIC unit. See Chapter 11, "Programmable Interrupt Controller (PIC)," for more information about the PIC unit.

When the PCI controller detects a read to the PCI CFG_DATA register, it checks the enable flag and the device number in the PCI CFG_ADDR register. If the enable bit is set, the bus number corresponds to the local PCI bus (bus number = 0x00), the device number is all ones (0b1_1111), the function number is all ones (0b111), and the register number is zero (0b00_0000), then the PCI controller performs an interrupt-acknowledge transaction. If the bus number indicates a nonlocal PCI bus, the PCI controller performs a type 1 configuration cycle translation, similar to any other configuration cycle for which the bus number does not match.

The address phase contains no valid information other than the interrupt-acknowledge command (PCI_C/$\overline{\text{BE}}$[3:0] = 0b0000). Although there is no explicit address, PCI_AD[31:0] are driven to a stable state, and parity is generated. Only one device (the system interrupt controller) on the PCI bus should respond to the interrupt-acknowledge command by asserting $\overline{\text{PCI\_DEVSEL}}$. All other devices on the bus should ignore the interrupt-acknowledge command. As stated previously, the device's PIC unit does not respond to PCI interrupt-acknowledge commands.

During the data phase, the responding device returns the interrupt vector on PCI_AD[31:0] when $\overline{\text{PCI\_TRDY}}$ is asserted. The size of the interrupt vector returned is indicated by the value driven on the PCI_C/$\overline{\text{BE}}$[3:0] signals.

The PCI controller also provides a direct way to generate PCI interrupt-acknowledge transactions. Reads from PCI INT_ACK at offset 0x008 generate PCI interrupt-acknowledge transactions. Note that processor writes to these addresses do nothing.

20.4.2.12.2 Special-Cycle Transactions

The special-cycle command provides a mechanism to broadcast select messages to all devices on the PCI bus. The special-cycle command contains no explicit destination address but is broadcast to all PCI agents.

When the PCI controller detects a write to PCI CFG_DATA, it checks the enable flag and the device number in PCI CFG_ADDR. If the enable bit is set, the bus number corresponds to the local PCI bus (bus number = 0x00), the device number is all ones (0b1_1111), the function number is all ones (0b111), and the register number is zero (0b00_0000), then the PCI controller performs a special-cycle transaction on the local PCI bus. If the bus number indicates a nonlocal PCI bus, the PCI controller performs a type 1 configuration cycle translation, similar to any other configuration cycle for which the bus number does not match.

Aside from the special-cycle command (PCI_C/$\overline{\text{BE}}$[3:0] = 0b0001) the address phase contains no other valid information. Although there is no explicit address, PCI_AD[31:0] are driven to a stable state, and parity is generated. During the data phase, PCI_AD[31:0] contain the special-cycle message and an optional data field. The special-cycle message is encoded on the 16 least-significant bits (PCI_AD[15:0]); the optional data field is encoded on the most-significant 16 lines (PCI_AD[31:16]). The special-cycle message encodings are assigned by the PCI SIG steering committee. The current list of defined encodings are provided in Table 20-51.

Table 20-51. Special-Cycle Message Encodings

PCI_AD[15:0]	Message
0x0000	SHUTDOWN
0x0001	HALT
0x0002	x86 architecture-specific
0x0003–0xFFFF	—

Note that the PCI controller does not automatically issue a special-cycle message when it enters any of its power-saving modes. It is the responsibility of software to issue the appropriate special-cycle message, if needed.

Each receiving agent must determine whether the special-cycle message is applicable to itself. Assertion of $\overline{\text{PCI\_DEVSEL}}$ in response to a special-cycle command is not necessary. The initiator of the special-cycle transaction can insert wait states but since there is no specific target, the special-cycle message and optional data field are valid on the first clock $\overline{\text{PCI\_IRDY}}$ is asserted. All special-cycle transactions are terminated by master-abort; however, the master-abort bit in the initiator's bus status register is not set for special-cycle terminations.

20.4.2.13 PCI Error Functions

PCI provides for parity and other system errors to be detected and reported. The PCI command register provides for selective enabling of specific PCI error detection. The PCI bus status register provides PCI error reporting. This section describes generation and detection of parity and error reporting for the PCI bus.

20.4.2.13.1 PCI Parity

Generating parity is not optional; it must be performed by all PCI-compatible devices. All PCI transactions, regardless of type, calculate even parity; that is, the number of ones on the PCI_AD[31:0], PCI_C/$\overline{\text{BE}}$[3:0], and PCI_PAR signals all sum to an even number.

Parity provides a way to determine, on each transaction, if the initiator successfully addressed the target and transferred valid data. The PCI_C/$\overline{\text{BE}}$[3:0] signals are included in the parity calculation to ensure that the correct bus command is performed (during the address phase) and correct data is transferred (during the data phase). The agent responsible for driving the bus must also drive even parity on the PAR and PCI_PAR64 signal one clock cycle after a valid address phase or valid data transfer, as shown in Figure 20-60.

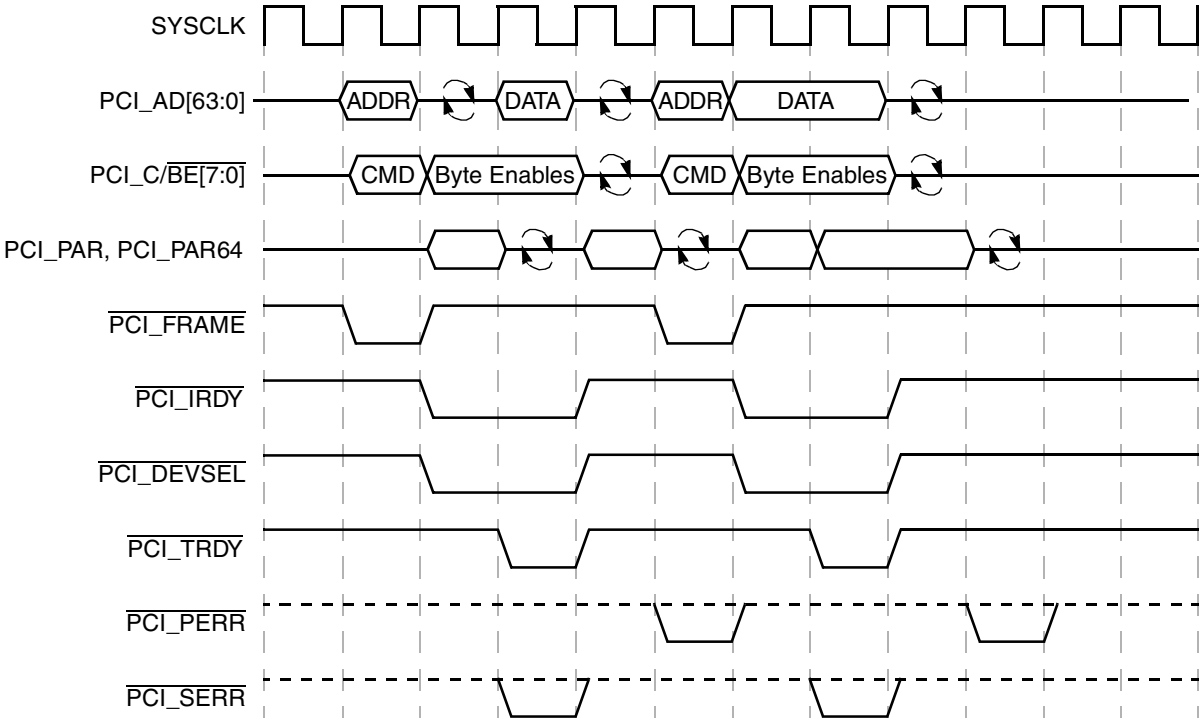


Figure 20-60. PCI Parity Operation

During the address and data phases, parity covers all 32 address/data signals and 4 command/byte enable signals, regardless of whether all lines carry meaningful information. Byte lanes not actually transferring data must contain stable (albeit meaningless) data and are included in parity calculation. During configuration, special-cycle, or interrupt-acknowledge commands; some address lines are not defined, but are driven to stable values and are included in parity calculation.

Agents that support parity checking must set the detected parity error bit in the PCI bus status register when a parity error is detected. Any additional response to a parity error is controlled by the parity error response bit in the PCI bus command register. If the parity error response bit is cleared, the agent ignores all parity errors.

20.4.2.13.2 Error Reporting

PCI provides for the detection and signaling of both parity and other system errors. Two signals are used to report these errors—$\overline{\text{PCI\_PERR}}$ and $\overline{\text{PCI\_SERR}}$. The $\overline{\text{PCI\_PERR}}$ signal is used exclusively to report data parity errors on all transactions except special cycles. The $\overline{\text{PCI\_SERR}}$ signal is used for other error signaling including address parity errors and data parity errors on special-cycle transactions; it may also be used to signal other system errors.

Table 20-52 shows the actions taken for each kind of error.

Table 20-52. PCI Mode Error Actions

PCI Error Type	Error Detect Register bit	PCI Status Register bit	Comment
PCI Outbound Read			
Received $\overline{\text{SERR}}$ at any phase	Rcvd $\overline{\text{SERR}}$	—	No data transferred
Received Parity Error for data phase	Mstr $\overline{\text{PERR}}$	Detected Parity Error, Master Data Parity Error Detected	No data transferred
Master Abort	Mstr abort	Received Master Abort	No data transferred
Target Abort	Trgt abort	Received Target Abort	No data transferred
Memory space violation	ORMSV	—	No data transferred. Only 8 bytes are requested in PCI bus
PCI Outbound Write			
Received $\overline{\text{SERR}}$ related to Address phase	Rcvd $\overline{\text{SERR}}$	—	May float AD bus to avoid contention
Received $\overline{\text{SERR}}$ related to Data phase	Rcvd $\overline{\text{SERR}}$	—	
Received $\overline{\text{PERR}}$ (Data phase)	Mstr $\overline{\text{PERR}}$	Master Data Parity Error	
Master Abort	Mstr abort	Received Master Abort	
Target Abort	Trgt abort	Received Target Abort	
Memory space violation	OWMSV	—	Only 8 bytes transferred.
PCI Inbound Read			
Detected Parity Error for Address phase	Addr Parity Error	Detected Parity Error, Signaled System Error	Float AD bus

Table 20-52. PCI Mode Error Actions (continued)

PCI Error Type	Error Detect Register bit	PCI Status Register bit	Comment
Detected Parity Error on upper address bus for Address phase (SAC or DAC)	—	—	
Received $\overline{\text{SERR}}$ at any phase	Received $\overline{\text{SERR}}$	—	
Received $\overline{\text{PERR}}$ (Data phase)	Target $\overline{\text{PERR}}$	—	
Internal error	Target Abort	Signaled Target Abort	
PCI Inbound Write			
Detected Parity Error for Address phase	Addr Parity Error	Detected Parity Error, Signaled System Error	Cache line purged
Detected Parity Error on upper address bus for Address phase (SAC or DAC)	—	—	
Received $\overline{\text{SERR}}$ at any phase	Rcvd $\overline{\text{SERR}}$	—	
Detected Parity Error for Data phase	Trgt $\overline{\text{PERR}}$	Detected Parity Error	Cache line purged

20.5 Initialization/Application Information

This section describes some tips for use of the PCI controller.

20.5.1 Power-On Reset Configuration Modes

The PCI controller can power-on in three modes: host mode, agent mode and agent configuration lock mode. Certain bits in the configuration registers are set differently according to the POR (power-on reset) mode. Also, certain configuration bits have different implications when compared with past Freescale parts and PCI implementations. Note that after reset, the device cannot be switched from one mode to another.

The affected configuration bits are defined in Table 20-53.

Table 20-53. Affected Configuration Register Bits for POR

Register (offset)	Bit	Name	Register Description
PCI Command Register (0x04)	2	Bus master	Controls whether the device can master a transaction on the PCI bus. If cleared, the device cannot master a transaction. This bit is independent of host or agent mode.
	1	Memory space	Controls the acknowledgement of inbound memory transactions. If cleared, all inbound memory accesses (including accesses to PCSRBAR space) end in a master abort. This bit is independent of host or agent mode.

Table 20-53. Affected Configuration Register Bits for POR (continued)

Register (offset)	Bit	Name	Register Description
PCI Bus Function Register (0x44)	5	ACL	Valid only in agent mode. Controls acknowledgement of inbound configuration accesses. If set, all inbound configuration accesses are retried. If cleared, inbound configuration accesses are acknowledged. In host mode all inbound configuration accesses end in master aborts.
	0	PAH	Determines whether the device is in agent or host mode. Zero indicates host mode.

The POR reset values for the affected configuration bits are described in Table 20-54.

Table 20-54. Power-On Reset Values for Affected Configuration Bits

Mode	Configuration Bit			
	Bus Master	Memory Space	ACL	PAH
Host	1	0	X	0
Agent	0	0	0	1
Agent configuration lock	0	0	1	1

20.5.1.1 Host Mode

When the device powers up in host mode, all inbound configuration accesses are ignored (and thus master aborted). The PBFR[ACL] bit is a don’t care. The device powers up with the ability to master transactions on the PCI bus, however in order to acknowledge memory transactions, the memory space bit must be set.

20.5.1.2 Agent Mode

When the device powers up in agent mode, it acknowledges inbound configuration accesses. However the device cannot master transactions or acknowledge inbound memory accesses on the PCI bus until the appropriate configuration bits (bus master and memory space, respectively) have been set.

20.5.1.3 Agent Configuration Lock Mode

Agent configuration lock mode is similar to agent mode with the added restriction that when the device powers up in agent configuration lock mode, it retries all inbound configuration accesses until the PBFR[ACL] bit is cleared. The purpose of this mode is to allow initial configuration on the port by the local processor before opening the port to be further configured by the external host. As in agent mode, the device in agent configuration lock mode cannot master transactions or acknowledge inbound memory accesses on the PCI bus until the appropriate configuration bits (bus master and memory space, respectively) have been set.

20.5.2 Byte Ordering

Whenever data must cross a bridge between two busses, the byte ordering of data on the source and destination buses must be considered. The internal platform bus of this device is inherently big endian and the PCI bus interface is inherently little endian.

There are two methods to handle ordering of data as it crosses a bridge—address invariance and data invariance. Address invariance preserves the addressing of bytes within a scalar data element, but not the relative significance of the bytes within that scalar. Conversely, data invariance preserves the relative significance of bytes within a scalar, but not the addressing of the individual bytes that make up a scalar.

This device uses address invariance as its byte ordering policy.

As stated above, address invariance preserves the byte address of each byte on an I/O interface as it is placed in memory or moved into a register. This policy can have the effect of reversing the significance order of bytes (most significant to least significant and vice versa), but it has the benefit of preserving the format of general data structures. Provided that software is aware of the endianness and format of the data structure, it can correctly interpret the data on either side of the bridge.

Figure 20-61 shows the transfer of a 4-byte scalar, 0x4142_4344, from a big endian source across an address invariant bridge to a little endian destination.

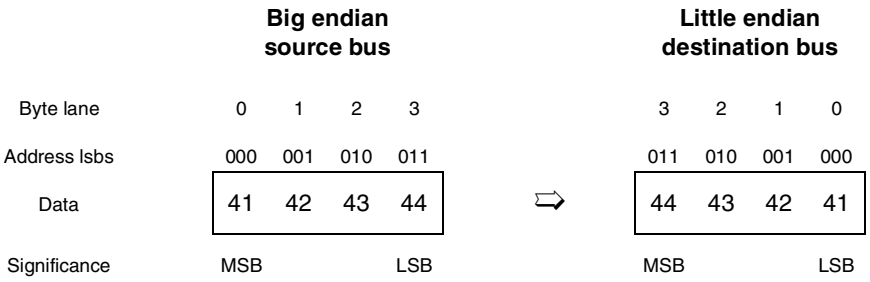


Figure 20-61. Address Invariant Byte Ordering—4 bytes Outbound

Note that although the significance of the bytes within the scalar have changed, the address of the individual bytes that make up the scalar have not changed. As long as software is aware that the source of the data used a big endian format, the data can be interpreted correctly.

Figure 20-63 shows data flowing the other way, from a little endian source to a big endian destination.

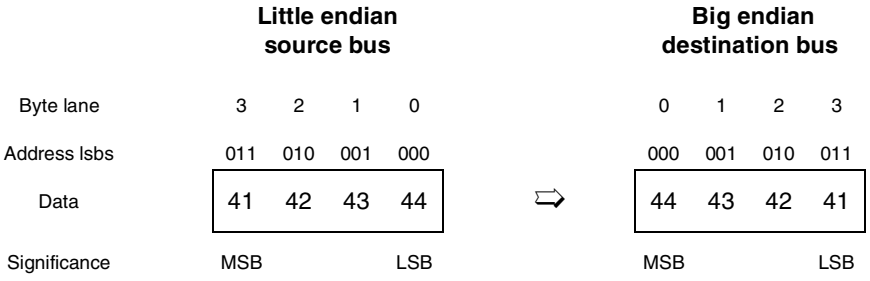


Figure 20-62. Address Invariant Byte Ordering—4 bytes Inbound

Figure 20-63 shows an outbound transfer of an 8-byte scalar, 0x5455_1617_CDCE_2728, using address invariance.

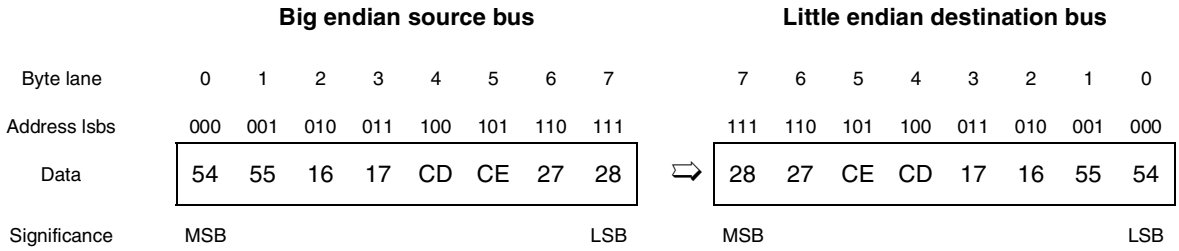


Figure 20-63. Address Invariant Byte Ordering—8 bytes Outbound

Figure 20-64 shows an inbound transfer of a 2-byte scalar, 0x5837, using address invariance.

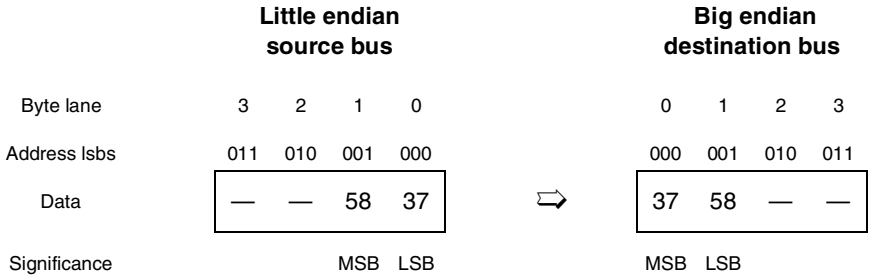


Figure 20-64. Address Invariant Byte Ordering—2 bytes Inbound

Note that in all of these examples, the original addresses of the individual bytes within the scalars (as created by the source) have been preserved.

20.5.2.1 Byte Order for Configuration Transactions

All internal memory-mapped registers in the CCSR space use big endian byte ordering. However, the PCI specification defines PCI configuration registers as little endian. All accesses to the PCI configuration port, CFG_DATA, including the those targeting the internal PCI configuration registers, use the address invariance policy as shown in Figure 20-65. Therefore, software must access CFG_DATA with little-endian formatted data—either using the **lwbrx**/**stwbrx** instructions or by manipulating the data before writing to and after reading from CFG_DATA.

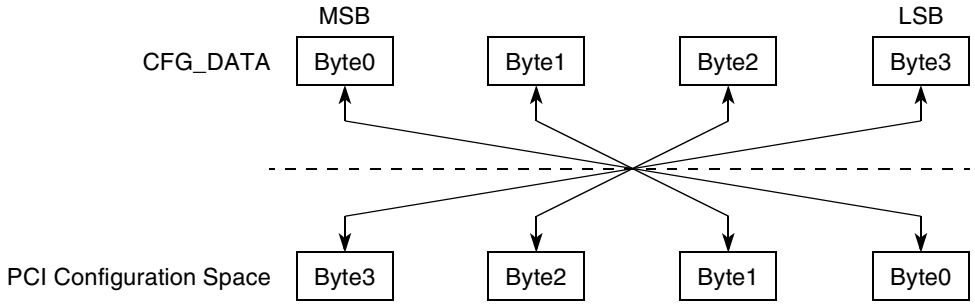


Figure 20-65. CFG_DATA Byte Ordering

Chapter 21
PCI Express Interface Controller

The PCI Express interface is compatible 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. Note that 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.

21.1 Introduction

The PCI Express controller provides the mechanism to communicate with PCI Express devices. Figure 21-1 is a high-level block diagram of the PCI Express controller.

21.1.1 Overview

The PCI Express controller connects the internal platform to a 2.5-GHz serial interface. The MPC8610 offers two instantiations of this controller yielding up to a ×4 link on SerDes port 1 and up to a ×8 link on SerDes port 2. The remainder of this chapter refers to a single PCI Express controller offering up to a x8 link interface. Notes are included to indicate variations for multiple instantiations.

As both an initiator and a target device, the PCI Express interface is capable of high-bandwidth data transfer and is designed to support next generation I/O devices. Upon coming out of reset, the PCI Express interface performs link width negotiation and exchanges flow control credits with its link partner. Once link autonegotiation is successful, the controller is in operation.

Internally, the design contains queues to keep track of inbound and outbound transactions. There is control logic that handles buffer management, bus protocol, transaction spawning and tag generation. In addition, there are memory blocks used to store inbound and outbound data.

The PCI Express controller can be configured to operate as either a PCI Express root complex (RC) or an endpoint (EP) device. An RC device connects the host CPU/memory subsystem to I/O devices while an

EP device typically denotes a peripheral or I/O device. In RC mode, a PCI Express type 1 configuration header is used; in EP mode, a PCI Express type 0 configuration header is used.

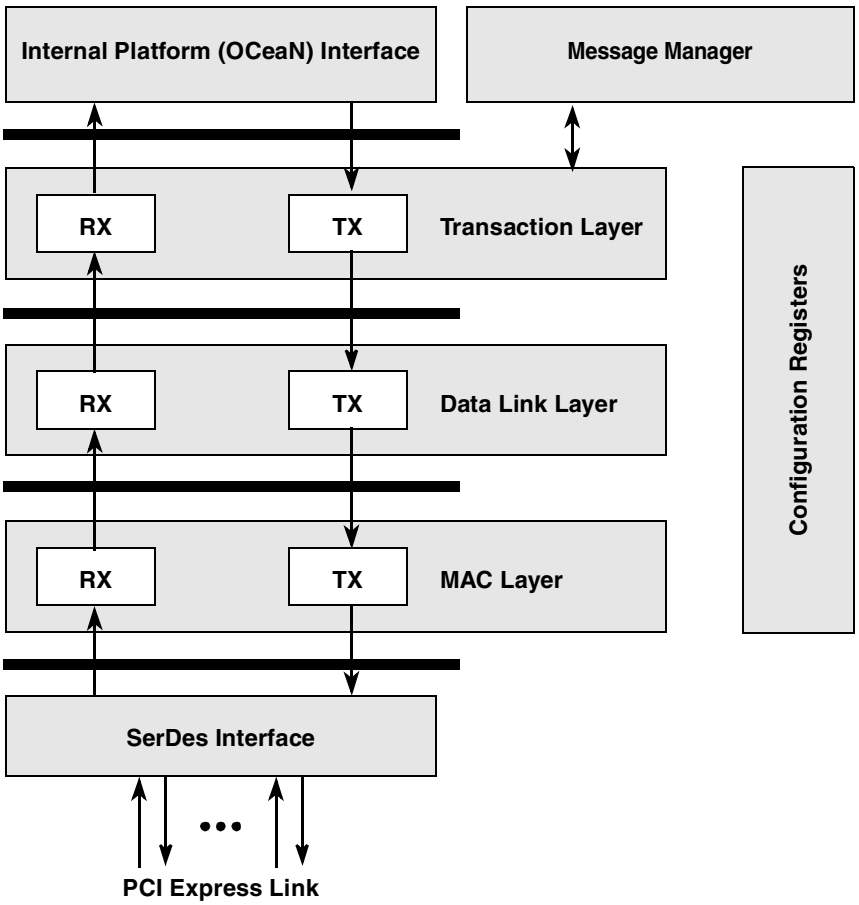


Figure 21-1. PCI Express Controller Block Diagram

As an initiator, the PCI Express controller supports memory read and write operations with a maximum transaction size of 256 bytes. In addition, configuration and I/O transactions are supported if the PCI Express controller is in RC mode. As a target interface, the PCI Express controller accepts read and write operations to local memory space. When configured as an EP device, the PCI Express controller accepts configuration transactions to the internal PCI Express configuration registers. Message generation and acceptance are supported in both RC and EP modes. Locked transactions and inbound I/O transactions are not supported.

21.1.1.1 Outbound Transactions

Outbound internal platform transactions to PCI Express are first mapped to a translation window to determine what PCI Express transactions are to be issued. A transaction from the internal platform can become a PCI Express Memory, I/O, Message, or Configuration transaction depending on the window attributes.

A transaction may be broken up into smaller sized transactions depending on the original request size, transaction type, and either the PCI Express device control register [MAX_PAYLOAD_SIZE] field for write requests or the PCI Express device control register [MAX_READ_SIZE] field for read requests. The controller performs PCI Express ordering rule checking to determine which transaction is to be sent on the PCI Express link.

In general, transactions are serviced in the order that they are received from the internal platform (OCeaN). Only when there is a stalled condition does the controller apply PCI Express ordering rules to outstanding transactions. For posted write transactions, once all data has been received from the internal platform (OCeaN), the data is forwarded to the PCI Express link and the transaction is considered as done. For non-posted write transactions, the controller waits for the completion packets to return before considering the transaction finished. For non-posted read transactions, the controller waits for all completion packets to return and then forwards all data back to the internal platform before terminating the transaction.

Note that after reset or when recovering from a link down condition, external transactions should not be attempted until the link has successfully trained. Software can poll the LTSSM state status register (PEX_LTSSM_STAT) to check the status of link training before issuing external requests.

21.1.1.2 Inbound Transactions

Inbound PCI Express transactions to internal platform are first mapped to a translation window to determine what internal platform transactions are to be issued.

A transaction may be broken up into smaller sized transactions when sending to the internal platform depending on the original request size, byte enables and starting/ending addresses. The controller performs PCI Express ordering rule checking to determine what transaction is to be sent next to the internal platform (OCeaN).

In general, transactions are serviced in the order that they are received from the PCI Express link. Only when there is a stalled condition does the controller apply PCI Express ordering to outstanding transactions. For posted write transactions, once all data has been received from the PCI Express link, the data is forwarded to the internal platform and the transaction is considered as done. For non-posted read transactions, the controller forwards internal platform data back to the PCI Express link.

Note that the controller splits transactions at the crossing of every 256-byte-aligned boundary when sending data back to the PCI Express link.

21.1.2 Features

The following is a list of features supported by the PCI Express controller:

- Compatible with the *PCI Express™ Base Specification, Revision 1.0a*
- Supports root complex (RC) and endpoint (EP) configurations
- 32- and 64-bit address support
- x8, x4, x2 and x1 link supported on Port 1; x4, x2 and x1 link supported on Port 2
- Supports accesses to all PCI Express memory and I/O address spaces (requestor only)
- Supports posting of processor-to-PCI Express and PCI Express-to-memory writes

- Supports strong and relaxed transaction ordering rules
- PCI Express configuration registers (type 0 in EP mode, type 1 in RC mode)
- Baseline and advanced error reporting support
- One virtual channel (VC0)
- 256-byte maximum payload size (MAX_PAYLOAD_SIZE)
- Supports three inbound general-purpose translation windows and one configuration window
- Supports four outbound translation windows and one default window
- Supports eight non-posted and four posted PCI Express transactions
- Supports up to six internal platform reads and eight internal platform writes. (The maximum number of outstanding transactions at any given time is eight.)
- Credit-based flow control management
- Supports PCI Express messages and interrupts
- Accepts up to 256-byte transactions from the internal platform (OCeaN)

21.1.3 Modes of Operation

There is one parameter that affects the mode of operation for the PCI Express controller. It is determined at power-on reset (POR) by a reset configuration signal as described in Chapter 4, "Reset, Clocking, and Initialization."

Table 21-1. POR Parameters for PCI Express Controller

Parameter	Description	Section/Page
Host/Agent Configuration	Selects between root complex (RC) and endpoint (EP) modes.	4.4.3.11/4-16

21.1.3.1 Root Complex/Endpoint Modes

The PCI Express controller can function as either a root complex (RC) or an endpoint (EP) on the PCI Express link. The host/agent configuration input signals cfg_host_agt[0:2] determine the RC/EP mode(See Section 4.4.3.11, "Host/Agent Configuration.")

21.2 External Signal Descriptions

PCI Express defines the connection between two devices as a link, which can be composed of a single 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 21-2 contains detailed descriptions of the external PCI Express interface signals. Note that there are two SD ports (SD1 and SD2) corresponding to PCI Express Controller 1 and 2.

Table 21-2. PCI Express Interface Signals—Detailed Signal Descriptions

Signal	I/O	Description	
SD1_RX[3:0] SD2_RX[7:0]	I	Receive data. The receive data signals carry PCI Express packet information. For PCI Express controller 1 (×4), SD1_RX[3:0] correspond to PCI Express RX lanes 3:0; for PCI Express controller 2 (×8), SD2_RX[7:0] correspond to PCI Express RX lanes 7:0. Note that lane reversal may affect the logical lane assignment. Refer to Section 21.4.1.3, "Lane Reversal," for more information.	
		State Meaning	Asserted/Negated—Represents data being received from the PCI Express interface.
		Timing	Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a*.
$\overline{\text{SD1\_RX}}$[3:0], $\overline{\text{SD2\_RX}}$[7:0]	I	Receive data, inverted. $\overline{\text{SD1\_RX}}$[3:0] and $\overline{\text{SD2\_RX}}$[7:0] are the inverted forms of the receive data signals ().	
		State Meaning	Asserted/Negated—Represents the inverse of data being received from the PCI Express interface.
		Timing	Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a*.
SD1_TX[3:0] SD2_TX[7:0]	O	Transmit data. The transmit data signals carry PCI Express packet information. For PCI Express controller 1 (×4), SD1_TX[3:0] correspond to PCI Express TX lanes 3:0; for PCI Express controller 2 (×8), SD2_TX[7:0] correspond to PCI Express TX lanes 7:0. Note that lane reversal may affect the logical lane assignment. Refer to Section 21.4.1.3, "Lane Reversal," for more information.	
		State Meaning	Asserted/Negated—Represents data being transmitted to the PCI Express interface.
		Timing	Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a*.
$\overline{\text{SD1\_TX}}$[3:0], $\overline{\text{SD2\_TX}}$[7:0]	O	Transmit data, inverted. $\overline{\text{SD1\_TX}}$[3:0] and $\overline{\text{SD2\_TX}}$[7:0] are the inverted form of the transmit data signals ().	
		State Meaning	Asserted/Negated—Represents the inverse of data being transmitted to the PCI Express interface.
		Timing	Assertion/Negation—As described in the *PCI Express Base Specification, Revision 1.0a*.

21.3 Memory Map/Register Definitions

The PCI Express interface supports the following register types:

- Memory-mapped registers—these registers control PCI Express address translation, PCI error management, and PCI Express configuration register access. These registers are described in Section 21.3.1, "PCI Express Memory Mapped Registers," and its subsections.
- PCI Express configuration registers contained within the PCI Express configuration space—these registers are specified by the PCI Express specification for every PCI Express device. These registers are described in Section 21.3.7, "PCI Express Configuration Space Access," and its subsections.

21.3.1 PCI Express Memory Mapped Registers

The PCI Express memory mapped registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR on the local side or the PEXCSRBAR on the PCI Express side) plus the block base address, plus the offset of the specific register to be accessed. Note that all

memory-mapped registers (except the PCI Express configuration data register, PEX_CONFIG_DATA) must only be accessed as 32-bit quantities.

Also note that although the table explicitly lists only the registers for the PCI Express controller 2 (×8), the register map for PCI Express controller 1 is the same except the for the block base address. Memory-mapped registers for PCI Express controller 2 (×8) begin at block base address 0x0_9000 and the memory-mapped registers for PCI Express controller 1 (×4) begin at block base address 0x0_A000.

Table 21-3 lists the memory-mapped registers. 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 21-3. PCI Express Memory-Mapped Register Map

PCI Express Controller 2 (×8) —Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
PCI Express Controller 2 (×8) Memory-Mapped Registers				
PCI Express Configuration Access Registers				
0x000	PEX_CONFIG_ADDR—PCI Express configuration address register	R/W	All zeros	21.3.2.1/21-10
0x004	PEX_CONFIG_DATA—PCI Express configuration data register	R/W	All zeros	21.3.2.2/21-10
0x008	Reserved	—	—	
0x00C	PEX_OTB_CPL_TOR—PCI Express outbound completion timeout register	R/W	0x0010_FFFF	21.3.2.3/21-11
0x010	PEX_CONF_RTY_TOR—PCI Express configuration retry timeout register	R/W	0x0400_FFFF	21.3.2.4/21-12
0x014	PEX_CONFIG—PCI Express configuration register	R/W	0x0000_0028	21.3.2.5/21-12
0x018–0x01C	Reserved	—	—	
PCI Express Power Management Event & Message Registers				
0x020	PEX_PME_MES_DR—PCI Express PME & message detect register	w1c	All zeros	21.3.3.1/21-13
0x024	PEX_PME_MES_DISR—PCI Express PME & message disable register	R/W	All zeros	21.3.3.2/21-15
0x028	PEX_PME_MES_IER—PCI Express PME & message interrupt enable register	R/W	All zeros	21.3.3.3/21-16
0x02C	PEX_PMCR—PCI Express power management command register	R/W	All zeros	21.3.3.4/21-18
0x030–0xBF4	Reserved	—	—	

Table 21-3. PCI Express Memory-Mapped Register Map (continued)

PCI Express Controller 2 (×8) —Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
PCI Express IP Block Revision Registers				
0xBF8	IP block revision register 1 (PEX_IP_BLK_REV1)	R	0x0208_0100	21.3.4.1/21-19
0xBFC	IP block revision register 2 (PEX_IP_BLK_REV2)	R	All zeros	21.3.4.2/21-19
PCI Express ATMU Registers				
Outbound Window 0 (Default)				
0xC00	PEXOTAR0—PCI Express outbound translation address register 0 (default)	R/W	All zeros	21.3.5.1.1/21-20
0xC04	PEXOTEAR0—PCI Express outbound translation extended address register 0 (default)	R/W	All zeros	21.3.5.1.2/21-21
0xC08–0xC0C	Reserved	—	—	
0xC10	PEXOWAR0—PCI Express outbound window attributes register 0 (default)	Mixed	0x8004_4023	21.3.5.1.4/21-22
0xC14–0xC1C	Reserved	—	—	
Outbound Window 1				
0xC20	PEXOTAR1—PCI Express outbound translation address register 1	R/W	All zeros	21.3.5.1.1/21-20
0xC24	PEXOTEAR1—PCI Express outbound translation extended address register 1	R/W	All zeros	21.3.5.1.2/21-21
0xC28	PEXOWBAR1—PCI Express outbound window base address register 1	R/W	All zeros	21.3.5.1.3/21-21
0xC2C	Reserved	—	—	
0xC30	PEXOWAR1—PCI Express outbound window attributes register 1	R/W	0x0004_4023	21.3.5.1.4/21-22
0xC34–0xC3C	Reserved	—	—	
Outbound Window 2				
0xC40	PEXOTAR2—PCI Express outbound translation address register 2	R/W	All zeros	21.3.5.1.1/21-20
0xC44	PEXOTEAR2—PCI Express outbound translation extended address register 2	R/W	All zeros	21.3.5.1.2/21-21
0xC48	PEXOWBAR2—PCI Express outbound window base address register 2	R/W	All zeros	21.3.5.1.3/21-21
0xC4C	Reserved	—	—	
0xC50	PEXOWAR2—PCI Express outbound window attributes register 2	R/W	0x0004_4023	21.3.5.1.4/21-22
0xC54–0xC5C	Reserved	—	—	
Outbound Window 3				
0xC60	PEXOTAR3—PCI Express outbound translation address register 3	R/W	All zeros[1]	21.3.5.1.1/21-20
0xC64	PEXOTEAR3—PCI Express outbound translation extended address register 3	R/W	All zeros	21.3.5.1.2/21-21

Table 21-3. PCI Express Memory-Mapped Register Map (continued)

PCI Express Controller 2 (×8) —Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xC68	PEXOWBAR3—PCI Express outbound window base address register 3	R/W	All zeros[2]	21.3.5.1.3/21-21
0xC6C	Reserved	—	—	
0xC70	PEXOWAR3—PCI Express outbound window attributes register 3	R/W	All zeros[3]	21.3.5.1.4/21-22
0xC74–0xC7C	Reserved	—	—	
Outbound Window 4				
0xC80	PEXOTAR4—PCI Express outbound translation address register 4	R/W	All zeros	21.3.5.1.1/21-20
0xC84	PEXOTEAR4—PCI Express outbound translation extended address register 4	R/W	All zeros	21.3.5.1.2/21-21
0xC88	PEXOWBAR4—PCI Express outbound window base address register 4	R/W	All zeros	21.3.5.1.3/21-21
0xC8C	Reserved	—	—	
0xC90	PEXOWAR4—PCI Express outbound window attributes register 4	R/W	0x0004_4023	21.3.5.1.4/21-22
0xC94–0xD9C	Reserved	—	—	
Inbound Window 3				
0xDA0	PEXITAR3—PCI Express inbound translation address register 3	R/W	All zeros	21.3.5.2.3/21-26
0xDA4	Reserved	—	—	
0xDA8	PEXIWBAR3—PCI Express inbound window base address register 3	R/W	All zeros	21.3.5.2.4/21-27
0xDAC	PEXIWBEAR3—PCI Express inbound window base extended address register 3	R/W	All zeros	21.3.5.2.5/21-27
0xDB0	PEXIWAR3—PCI Express inbound window attributes register 3	R/W	0x20F4_4023	21.3.5.2.6/21-28
0xDB4–0xDBC	Reserved	—	—	
Inbound Window 2				
0xDC0	PEXITAR2—PCI Express inbound translation address register 2	R/W	All zeros	21.3.5.2.3/21-26
0xDC4	Reserved	—	—	
0xDC8	PEXIWBAR2—PCI Express inbound window base address register 2	R/W	All zeros	21.3.5.2.4/21-27
0xDCC	PEXIWBEAR2—PCI Express inbound window base extended address register 2	R/W	All zeros	21.3.5.2.5/21-27
0xDD0	PEXIWAR2—PCI Express inbound window attributes register 2	R/W	0x20F4_4023	21.3.5.2.6/21-28
0xDD4–0xDDC	Reserved	—	—	
Inbound Window 1				
0xDE0	PEXITAR1—PCI Express inbound translation address register 1	R/W	All zeros	21.3.5.2.3/21-26
0xDE4	Reserved	—	—	
0xDE8	PEXIWBAR1—PCI Express inbound window base address register 1	R/W	All zeros	21.3.5.2.4/21-27

Table 21-3. PCI Express Memory-Mapped Register Map (continued)

PCI Express Controller 2 (×8) —Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xDEC	Reserved	—	—	
0xDF0	PEXIWAR1—PCI Express inbound window attributes register 1	R/W	0x20F4_4023	21.3.5.2.6/21-28
0xDF4–0xDFC	Reserved	—	—	
PCI Express Error Management Registers				
0xE00	PEX_ERR_DR—PCI Express error detect register	w1c	All zeros	21.3.6.1/21-30
0xE04	Reserved	—	—	—
0xE08	PEX_ERR_EN—PCI Express error interrupt enable register	R/W	All zeros	21.3.6.2/21-33
0xE0C	Reserved	—	—	—
0xE10	PEX_ERR_DISR—PCI Express error disable register	R/W	All zeros	21.3.6.3/21-35
0xE14–0xE1C	Reserved	—	—	—
0xE20	PEX_ERR_CAP_STAT—PCI Express error capture status register	Mixed	All zeros	21.3.6.4/21-36
0xE24	Reserved	—	—	—
0xE28	PEX_ERR_CAP_R0—PCI Express error capture register 0	R/W	All zeros	21.3.6.5/21-37
0xE2C	PEX_ERR_CAP_R1—PCI Express error capture register 1	R/W	All zeros	21.3.6.6/21-39
0xE30	PEX_ERR_CAP_R2—PCI Express error capture register 2	R/W	All zeros	21.3.6.7/21-40
0xE34	PEX_ERR_CAP_R3—PCI Express error capture register 3	R/W	All zeros	21.3.6.8/21-42
0xE38–0xFFC	Reserved	—	—	
PCI Express Controller 1 (×4) Memory-Mapped Registers				
0x000–0xFFC	PCI Express Controller 1 (×4) registers **Note:** All registers defined for PCI Express Controller 2 (×8) are also defined for PCI Express Controller 1 (×4); the offsets of PCI Express Controller 1 (×4) registers are the same except they have a different block base address.			

1 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOTAR3 is 0x000F_FFF0.

2 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOWBAR3 is 0x000F_FF00.

3 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOWAR3 is 0x8004_400F.

21.3.2 PCI Express Configuration Access Registers

21.3.2.1 PCI Express Configuration Address Register (PEX_CONFIG_ADDR)

The PCI Express configuration address register, shown in Figure 21-2, contains address information for accesses to PCI Express internal and external configuration registers.

Offset 0x000 Access: Read/Write

	0	1–3	4–7	8–15	16–20	21–23	24–29	30–31
R W	EN	—	EXTREGN	BUSN	DEVN	FUNCN	REGN	—
Reset	All zeros							

Figure 21-2. PCI Express Configuration Address Register (PEX_CONFIG_ADDR)

The fields of the PCI Express configuration address register are described in Table 21-4.

Table 21-4. PEX_CONFIG_ADDR Field Descriptions

Bits	Name	Description
0	EN	Enable. This bit allows a PCI Express configuration access when PEX_CONFIG_DATA is accessed. If this bit is cleared, writing to PEX_CONFIG_DATA has no effect and reading PEX_CONFIG_DATA returns unknown data.
1–3	—	Reserved
4–7	EXTREGN	Extended register number. This field allows access to extended PCI Express configuration space (that is, the registers in the offset range from 0x100 to 0xFFF).
8–15	BUSN	Bus number. PCI bus number to access
16–20	DEVN	Device number. Device number to access on specified bus
21–23	FUNCN	Function number. Function to access within specified device
24–29	REGN	Register number. 32-bit register to access within specified device
30–31	—	Reserved

Both root complex (RC) and endpoint (EP) configuration headers contain 4096 bytes of address space. To access a register within the header, both the extended register number and the register number fields are concatenated to form the 4-byte aligned address of the register. That is, the register address is extended register number 0b00.

21.3.2.2 PCI Express Configuration Data Register (PEX_CONFIG_DATA)

The PCI Express configuration data register, show in Figure 21-3, is a 32-bit port for internal and external configuration access. Note that accesses of 1, 2, or 4 bytes to the PCI Express configuration data register

are allowed. Also note that accesses to the little-endian PCI Express configuration space must be properly formatted. See Section 21.4.1.2.1, “Byte Order for Configuration Transactions,” for more information.

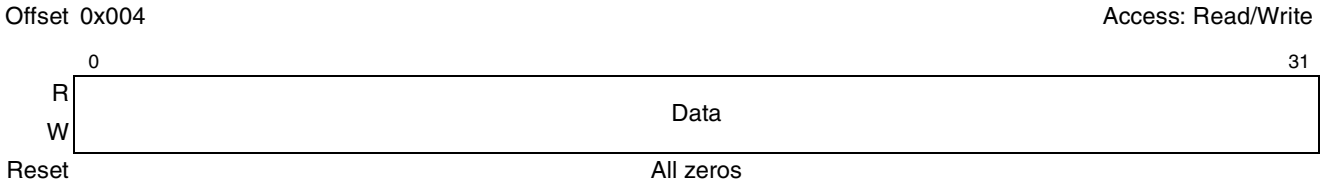


Figure 21-3. PCI Express Configuration Data Register (PEX_CONFIG_DATA)

The fields of the PCI Express configuration data register are described in Table 21-5.

Table 21-5. PEX_CONFIG_DATA Field Descriptions

Bits	Name	Description
0–31	Data	A read or write to this register starts a PCI Express configuration cycle if the PEX_CONFIG_ADDR enable bit is set (PEX_CONFIG_ADDR[EN] = 1).

21.3.2.3 PCI Express Outbound Completion Timeout Register (PEX_OTB_CPL_TOR)

The PCI Express outbound completion timeout register, shown in Figure 21-4, contains the maximum wait time for a response to come back as a result of an outbound non-posted request before a timeout condition occurs.

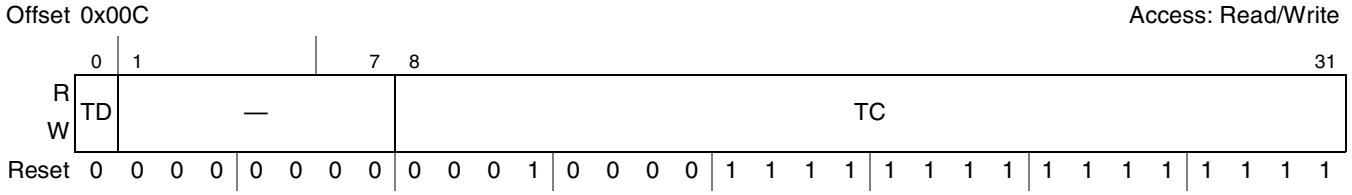


Figure 21-4. PCI Express Outbound Completion Timeout Register (PEX_OTB_CPL_TOR)

The fields of the PCI Express outbound completion timeout register are described in Table 21-6.

Table 21-6. PEX_OTB_CPL_TOR Field Descriptions

Bits	Name	Description
0	TD	Timeout disable. This bit controls the enabling/disabling of the timeout function. 0 Enable completion timeout 1 Disable completion timeout
1–7	—	Reserved
8–31	TC	Timeout counter. This is the value that is used to load the response counter of the completion timeout. One TC unit is 8× the PCI Express controller clock period; that is, one TC unit is 20 ns at 400 MHz, and 30 ns at 266.66 MHz. The following are examples of timeout periods based on different TC settings: 0x00_0000Reserved 0x10_FFFF22.28 ms at 400 MHz controller clock; 33.34 ms at 266.66 MHz controller clock 0xFF_FFFF335.54 ms at 400 MHz controller clock; 503.31 ms at 266.66 MHz controller clock

21.3.2.4 PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR)

The PCI Express configuration retry timeout register, shown in Figure 21-5, contains the maximum time period during which retries of configuration transactions which resulted in a CRS response occur.

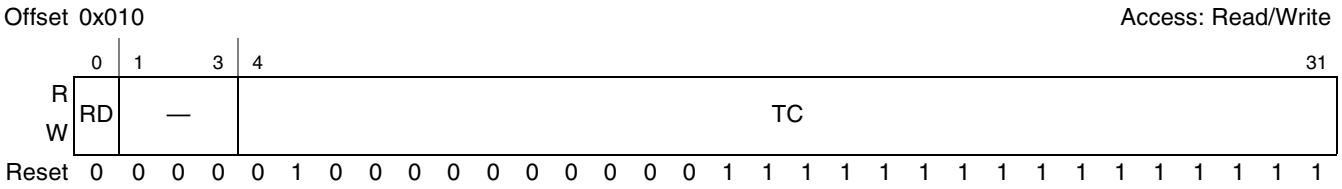


Figure 21-5. PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR)

The fields of the PCI Express configuration retry timeout register are described in Table 21-7.

Table 21-7. PEX_CONF_RTY_TOR Field Descriptions

Bits	Name	Description
0	RD	Retry disable. This bit disables the retry of a configuration transaction that receives a CRS status response packet. 0 Enable retry of a configuration transaction in response to receiving a CRS status response until the timeout counter (defined by the PEX_CONF_RTY_TOR[TC] field) has expired. 1 Disable retry of a configuration transaction regardless of receiving a CRS status response.
1–3	—	Reserved
4–31	TC	Timeout counter. This is the value that is used to load the CRS response counter. One TC unit is 8× the PCI Express controller clock period; that is, one TC unit is 20 ns at 400 MHz and 30 ns at 266.66 MHz. Timeout period based on different TC settings: 0x000_0000 Reserved 0x400_FFFF 1.34 s at 400 MHz controller clock, 2.02 s at 266.66 MHz controller clock 0xFFF_FFFF 5.37 s at 400 MHz controller clock, 8.05 s at 266.66 MHz controller clock

21.3.2.5 PCI Express Configuration Register (PEX_CONFIG)

The PCI Express configuration register, shown in Figure 21-6, contains various control switches for the controller.

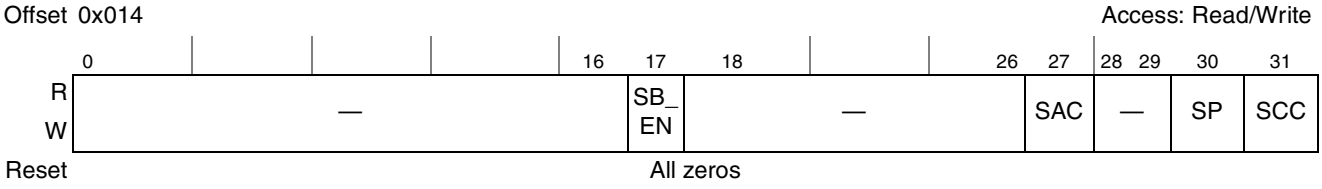


Figure 21-6. PCI Express Configuration Register (PEX_CONFIG)

The fields of the PCI Express configuration register are described in Table 21-8.

Table 21-8. PEX_CONFIG Field Descriptions

Bits	Name	Description
0–16	—	Reserved
17	SB_EN	Southbridge enable. Enables the ability to operate with certain southbridge devices that require being configured on bus 0. Note that this bit is for use in RC mode only; this bit must be cleared in EP mode. 0 Configuration transactions targeting bus 0 are handled internally and are not allowed to pass to the external link. 1 Configuration transactions targeting bus 0 are allowed to pass to the external link.
18–26	—	Reserved
27	SAC	Sense ASPM Control. This bit controls the default value of ASPM of PEX Link Control Register's bit 0. See Section 21.3.9.11, "PCI Express Link Control Register—0x5C," for more information.
28–29	—	Reserved
30	SP	Slot Present. This bit controls the default value of the PCI Express capabilities register [slot] bit. See Section 21.3.9.6, "PCI Express Capabilities Register—0x4E," for more information.
31	SCC	Slot Clock Configuration. This bit controls the default value of the PCI Express link status register [SCC] bit. See Section 21.3.9.12, "PCI Express Link Status Register—0x5E," for more information.

21.3.3 PCI Express Power Management Event and Message Registers

21.3.3.1 PCI Express PME and Message Detect Register (PEX_PME_MES_DR)

The PCI Express PME and message detect register, shown in Figure 21-7, logs inbound messages and PME events that are detected by the PCI Express controller. This register is a write-1-to-clear type register.

Offset 0x020 Access: w1c

	0	15
R	—	
W		
Reset	All zeros	

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PTO	—	ENL23	EXL23	—	HRD	LDD	—		AION	AIB	AIOF	PION	PIB	PIOF	ABP
W	w1c		w1c	w1c		w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	All zeros															

Figure 21-7. PCI Express PME and Message Detect Register (PEX_PME_MES_DR)

The fields of the PCI Express PME and message detect register are described in Table 21-9.

Table 21-9. PEX_PME_MES_DR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	PTO	PME turn off. This bit indicates the detection of a PME_Turn_Off message. This bit is only valid in EP mode. 1 A PME_Turn_Off_message is detected 0 No PME_Turn_Off message detected
17	—	Reserved. Note that during normal operation, this bit may be set (falsely). The bit may be ignored and cleared (w1c) without consequence.
18	ENL23	Entered L2/L3 ready state. This bit indicates that the PCI Express controller has entered L2/L3 state. This is only valid in RC mode. 1 L2/L3 ready state has been entered 0 L2/L3 ready state has not been entered
19	EXL23	Exit L2/L3 ready state. This bit indicates that the PCI Express controller has exited the L2/L3 state. This is only valid in RC mode. 1 Exit L2/L3 state has been detected 0 Exit L2/L3 state not detected
20	—	Reserved. Note that during normal operation, this bit may be set (falsely). The bit may be ignored and cleared (w1c) without consequence.
21	HRD	Hot reset detected. This bit indicates that the PCI Express controller has detected a hot reset condition on the link. The controller is reset and cleans up all outstanding transactions. Link retraining takes place once hot reset state is exited. This is valid only in EP mode. 1 Hot reset request has been detected 0 Hot reset request not detected
22	LDD	Link down detected. This bit indicates that a link down condition has been detected. The controller is reset and then cleans up all outstanding transactions. Link retraining takes place once the controller has cleaned itself up. Note that for EP, this bit and HRD are typically set when a hot reset event is detected. 1 Link down has been detected 0 Link down not detected
23–24	—	Reserved
25	AION	Attention indicator on. This bit indicates the detection of an Attention_Indicator_On message. This bit is only valid in EP mode. 1 Attention indicator on message is detected 0 No attention indicator on message detected
26	AIB	Attention indicator blink. This bit indicates the detection of an Attention_Indicator_Blink message. This bit is only valid in EP mode. 1 Attention indicator blink message is detected 0 No attention indicator blink message detected
27	AIOF	Attention indicator off. This bit indicates the detection of an Attention_Indicator_Off message. This bit is only valid in EP mode. 1 Attention indicator off message is detected 0 No attention indicator off message detected
28	PION	Power indicator on. This bit indicates the detection of a Power_Indicator_On message. This bit is only valid in EP mode. 1 Power indicator on message is detected 0 No power indicator on message detected

Table 21-9. PEX_PME_MES_DR Field Descriptions (continued)

Bits	Name	Description
29	PIB	Power indicator blink. This bit indicates the detection of an Power_Indicator_Blink message. This bit is only valid in EP mode. 1 Power indicator blink message is detected 0 No power indicator blink message detected
30	PIOF	Power indicator off. This bit indicates the detection of an Power_Indicator_Off message. This bit is only valid in EP mode. 1 Power indicator off message is detected 0 No power indicator off message detected
31	ABP	Attention button pressed. This bit indicates the detection of an Attention_Button_Pressed message. This bit is only valid in RC mode. 1 Attention button press message is detected 0 No attention button press message detected

21.3.3.2 PCI Express PME and Message Disable Register (PEX_PME_MES_DISR)

The PCI Express PME and message disable register, shown in Figure 21-8, when set, prevents the detection of the corresponding bits in the PCI Express PME and message detect register.

Offset 0x024 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	PTOD	—	ENL23D	EXL23D	—	HRDD	LDDD	—	AIONDD	AIBD	AIOFD	PIOND	PIBD	PIOFD	ABPD
W															
Reset	All zeros														

Figure 21-8. PCI Express PME and Message Disable Register (PEX_PME_MES_DISR)

The fields of the PCI Express PME and message disable register are described in Table 21-10.

Table 21-10. PEX_PME_MES_DISR Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	PTOD	PME turn off disable. When set, this bit disables the setting of PEX_PME_MES_DR[PTO] bit. 1 Disable PME_Turn_Off_message detection 0 Enable PME_Turn_Off message detection
17	—	Reserved
18	ENL23D	Entered_L2/L3 ready disable. When set, this bit disables the setting of PEX_PME_MES_DR[ENL23] bit. 1 Disable Entered_L2/L3 ready state detection 0 Enable Entered_L2/L3 ready state detection
19	EXL23D	Exited_L2/L3 ready disable. When set, this bit disables the setting of PEX_PME_MES_DR[EXL23] bit. 1 Disable Exited_L2/L3 ready state detection 0 Enable Exited_L2/L3 ready state detection

Table 21-10. PEX_PME_MES_DISR Field Descriptions (continued)

Bits	Name	Description
20	—	Reserved
21	HRDD	Hot reset detected disable. When set, this bit disables the setting of PEX_PME_MES_DR[HRD] bit. 1 Disable hot reset state detection 0 Enable hot reset state detection
22	LDDD	Link down detected disable. When set, this bit disables the setting of PEX_PME_MES_DR[LDD] bit. 1 Disable link down state detection 0 Enable link down state detection
23–24	—	Reserved
25	AIOND	Attention indicator on disable. When set, this bit disables the setting of PEX_PME_MES_DR[AION] bit. 1 Disable attention indicator on message detection 0 Enable attention indicator on message detection
26	AIBD	Attention indicator blink disable. When set, this bit disables the setting of PEX_PME_MES_DR[AIB] bit. 1 Disable attention indicator blink message detection 0 Enable attention indicator blink message detection
27	AIOFD	Attention indicator off disable. When set, this bit disables the setting of PEX_PME_MES_DR[AIOF] bit. 1 Disable attention indicator off message detection 0 Enable attention indicator off message detection
28	PIOND	Power indicator on disable. When set, this bit disables the setting of PEX_PME_MES_DR[PION] bit. 1 Disable power indicator on message detection 0 Enable power indicator on message detection
29	PIBD	Power indicator blink disable. When set, this bit disables the setting of PEX_PME_MES_DR[PIB] bit. 1 Disable power indicator blink message detection 0 Enable power indicator blink message detection
30	PIOFD	Power indicator off disable. When set, this bit disables the setting of PEX_PME_MES_DR[PIOF] bit. 1 Disable power indicator off message detection 0 Enable power indicator off message detection
31	ABPD	Attention button pressed disable. When set, this bit disables the setting of PEX_PME_MES_DR[ABP] bit. 1 Disable attention button press message detection 0 Enable attention button press message detection

21.3.3.3 PCI Express PME and Message Interrupt Enable Register (PEX_PME_MES_IER)

The PCI Express PME and message interrupt enable register, shown in Figure 21-9, allows for the detection of a message or a PME event to generate an interrupt, provided that the corresponding bit in the PCI Express PME and message detect register is set.

Offset 0x028 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	PTO IE	—	ENL23 IE	EXL23 IE	—	HRD IE	LDD IE	—	AION IE	AIB IE	AIOF IE	PION IE	PIB IE	PIOF IE	ABP IE
Reset	All zeros														

Figure 21-9. PCI Express PME and Message Interrupt Enable Register (PEX_PME_MES_IER)

Table 21-11 shows the fields of the PCI Express PME and message interrupt enable register.

Table 21-11. PEX_PME_MES_IER Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16	PTOIE	PME turn off interrupt enable. When set and PEX_PME_MES_DR[PTO]=1 generates an interrupt. 1 Enable PME_Turn_Off_message interrupt generation 0 Disable PME_Turn_Off message interrupt generation
17	—	Reserved
18	ENL23IE	Entered L2/L3 ready interrupt enable. When set and PEX_PME_MES_DR[ENL23]=1 generates an interrupt. 1 Enable Entered_L2/L3 ready state interrupt generation 0 Disable Entered_L2/L3 ready state interrupt generation
19	EXL23IE	Exited L2/L3 ready interrupt enable. When set and PEX_PME_MES_DR[EXL23]=1 generates an interrupt. 1 Enable Exited_L2/L3 ready state interrupt generation 0 Disable Exited_L2/L3 ready state interrupt generation
20	—	Reserved
21	HRDIE	Hot reset detected interrupt enable. When set and PEX_PME_MES_DR[HRD]=1 generates an interrupt. 1 Enable hot reset state interrupt generation 0 Disable hot reset state interrupt generation
22	LDDIE	Link down detected interrupt enable. When set and PEX_PME_MES_DR[LDD]=1 generates an interrupt. 1 Enable link down state interrupt generation 0 Disable link down state interrupt generation
23–24	—	Reserved
25	AIONIE	Attention indicator on interrupt enable. When set and PEX_PME_MES_DR[AION]=1 generates an interrupt. 1 Enable attention indicator on message interrupt generation 0 Disable attention indicator on message interrupt generation
26	AIBIE	Attention indicator blink interrupt enable. When set and PEX_PME_MES_DR[AIB]=1 generates an interrupt. 1 Enable attention indicator blink message interrupt generation 0 Disable attention indicator blink message interrupt generation

Table 21-11. PEX_PME_MES_IER Field Descriptions (continued)

Bits	Name	Description
27	AIOFIE	Attention indicator off interrupt enable. When set and PEX_PME_MES_DR[AIOF]=1 generates an interrupt. 1 Enable attention indicator off message interrupt generation 0 Disable attention indicator off message interrupt generation
28	PIONIE	Power indicator on interrupt enable. When set and PEX_PME_MES_DR[PION]=1 generates an interrupt. 1 Enable power indicator on message interrupt generation 0 Disable power indicator on message interrupt generation
29	PIBIE	Power indicator blink interrupt enable. When set and PEX_PME_MES_DR[PIB]=1 generates an interrupt. 1 Enable power indicator blink message interrupt generation 0 Disable power indicator blink message interrupt generation
30	PIOFIE	Power indicator off interrupt enable. When set and PEX_PME_MES_DR[PIOF]=1 generates an interrupt. 1 Enable power indicator off message interrupt generation 0 Disable power indicator off message interrupt generation
31	ABPIE	Attention button pressed interrupt enable. When set and PEX_PME_MES_DR[ABP]=1 generates an interrupt. 1 Enable attention button press message interrupt generation 0 Disable attention button press message interrupt generation

21.3.3.4 PCI Express Power Management Command Register (PEX_PMCR)

The PCI Express power management command register, shown in Figure 21-10, provides software a mechanism to allow the PCI Express controller to get back to L0 link state.

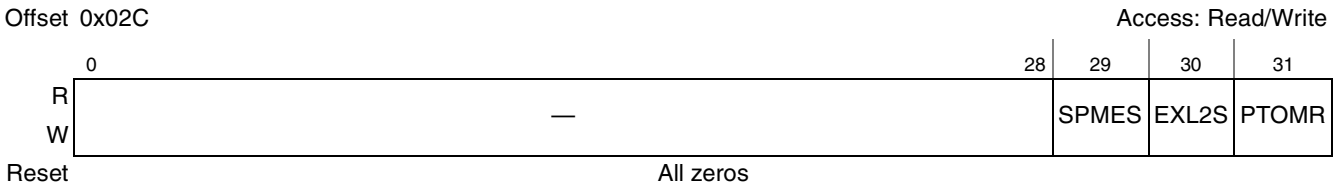


Figure 21-10. PCI Express Power Management Command Register (PEX_PMCR)

The fields of the PCI Express power management command register are described in Table 21-12.

Table 21-12. PEX_PMCR Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29	SPMES	Set PME status. This sets the PME status bit and if PME is enabled (see Section 21.3.9.3, “PCI Express Power Management Status and Control Register—0x48,” for more information) it transmits a PM_PME message upstream. This bit should not be used when in RC mode. This bit is self-clearing.
30	EXL2S	Exit L2 state. When set exits the link state out of L2/L3 ready state in order to send new requests. The request is only made when entered_L2/L3 ready state is active. This bit is self-clearing. When the link has exited L2/L3 ready state, the status bit Exit_L2/L3 ready state is set. This bit should not be used when in EP mode.
31	PTOMR	PME_Turn_Off message request. When set broadcasts a PME turn_off message. This bit should not be used when in EP mode. This bit is self-clearing

21.3.4 PCI Express IP Block Revision Registers

21.3.4.1 IP Block Revision Register 1 (PEX_IP_BLK_REV1)

The IP block revision register 1 is shown in Figure 21-11.

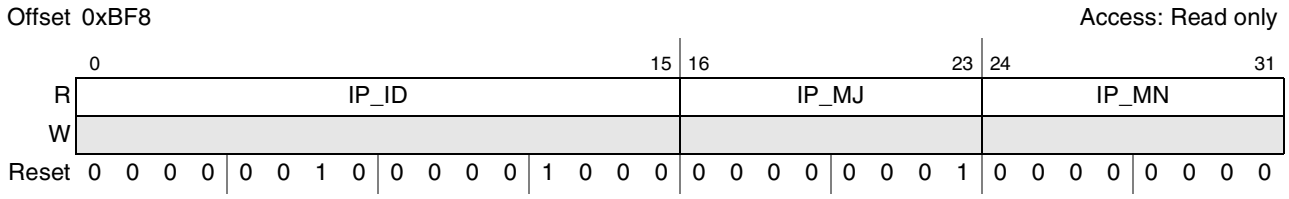


Figure 21-11. IP Block Revision Register 1

Table 21-13 contains descriptions of the fields of the IP block revision register 1.

Table 21-13. PCI Express IP Block Revision Register 1 Field Descriptions

Bits	Name	Description
0–15	IP_ID	Block ID
16–23	IP_MJ	Block Major Revision
24–31	IP_MN	Block Minor Revision

21.3.4.2 IP Block Revision Register 2 (PEX_IP_BLK_REV2)

The IP block revision register 2 is shown in Figure 21-12.

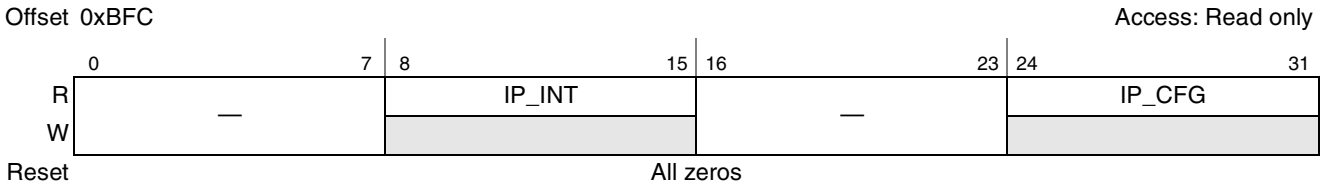


Figure 21-12. IP Block Revision Register 2

Table 21-14 contains descriptions of the fields of the IP block revision register 2.

Table 21-14. PCI Express IP Block Revision Register 2 Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–15	IP_INT	Block integration option
16–23	—	Reserved
24–31	IP_CFG	Block configuration option

21.3.5 PCI Express ATMU Registers

21.3.5.1 PCI Express Outbound ATMU Registers

The outbound address translation windows must be aligned based on the granularity selected by the size fields. Outbound window misses use the default outbound register set (outbound ATMU window 0). Overlapping outbound windows (1–4) are not supported and will cause undefined behavior. Note that for RC mode, all outbound transactions post ATMU must hit either into the memory base/limit range or the prefetchable memory base/limit range defined in the PCI Express type 1 header. For EP mode, there is no such requirement.

Note that in RC mode, there is no checking on whether the translated address actually hits into the memory base/limit range. It will just pass it through as is.

Figure 21-13 shows the outbound transaction flow.

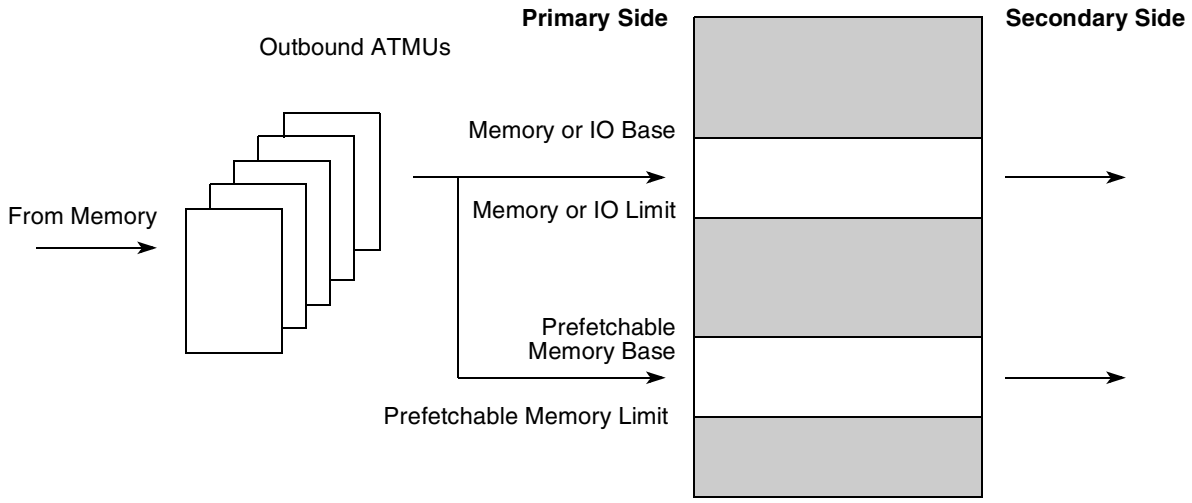


Figure 21-13. RC Outbound Transaction Flow

21.3.5.1.1 PCI Express Outbound Translation Address Registers (PEXOTAR*n*)

The PCI Express outbound translation address registers, shown in Figure 21-14, select the starting addresses in the system address space for window hits within the PCI Express outbound address translation windows. The new translated address is created by concatenating the transaction offset to this translation address.

Offset Window 0: 0xC00, Window 1: 0xC20 Access: Read/Write
Window 2: 0xC40, Window 3: 0xC60, Window 4: 0xC80

	0	11	12	31
R	TEA		TA	
W				
Reset	All zeros[1]			

Figure 21-14. PCI Express Outbound Translation Address Registers (PEXOTAR*n*)

[1] If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOTAR3 is 0x000F_FFF0.

Table 21-15 describes the fields of the PCI Express outbound translation address registers.

Table 21-15. PEXOTAR*n* Field Descriptions

Bits	Name	Description
0–11	TEA	Translation extended address. System address which indicates the starting point of the outbound translated address. The translation address must be aligned based on the size field. Corresponds to PCI Express address bits [43:32] (bit 32 is the lsb).
12–31	TA	Translation address. System address which indicates the starting point of the outbound translated address. The translation address must be aligned based on the size field. This corresponds to PCI Express address bits [31:12].

21.3.5.1.2 PCI Express Outbound Translation Extended Address Registers (PEXOTEAR*n*)

The PCI Express outbound translation extended address registers, shown in Figure 21-15, contain the most-significant bits of a 64 bit translation address.

Offset Window 0: 0xC04, Window 1: 0xC24 Access: Read/Write
Window 2: 0xC44, Window 3: 0xC64, Window 4: 0xC84

	0	11	12	31
R	—		TEA	
W				
Reset	All zeros			

Figure 21-15. PCI Express Outbound Translation Extended Address Registers (PEXOTEAR*n*)

Table 21-16 describes the fields of the PCI Express outbound translation extended address registers.

Table 21-16. PCI Express Outbound Extended Address Translation Register *n* Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	TEA	Translation extended address. System address which indicates the starting point of the outbound translated address. The translation address must be aligned based on the size field. Corresponds to PCI Express address bits [63:44].

21.3.5.1.3 PCI Express Outbound Window Base Address Registers (PEXOWBAR*n*)

The PCI Express outbound window base address registers, shown in Figure 21-16, select the base address for the windows which are translated to the external address space. Addresses for outbound transactions are compared to these windows. If such a transaction does not fall within one of these spaces the transaction is forwarded through a default register set.

Offset Window 1: 0xC28, Window 2: 0xC48 Access: Read/Write
Window 3: 0xC68, Window 4: 0xC88

	0–7	8–11	12–31
R W	—	WBEA	WBA
Reset			All zeros[1]

Figure 21-16. PCI Express Outbound Window Base Address Registers (PEXOWBAR*n*)

1 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOWBAR3 is 0x000F_FF00.

Table 21-17 describes the fields of the PCI Express outbound window base address registers.

Table 21-17. PCI Express Outbound Window Base Address Register *n* Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–11	WBEA	Window base extended address. Source address which is the starting point for the outbound translation window. The window must be aligned based on the size selected in the window size bits. Correspond to internal platform address bits [0:3]. (where 0 is the msb of the internal platform address)
12–31	WBA	Window base address. Source address which is the starting point for the outbound translation window. The window must be aligned based on the size selected in the window size bits. This corresponds to internal platform address bits [4:23].

21.3.5.1.4 PCI Express Outbound Window Attributes Registers (PEXOWAR*n*)

The PCI Express outbound window attributes registers, shown in Figure 21-17 and Figure 21-18, define the window sizes to translate and other attributes for the translations. 64 Gbytes is the largest window size allowed. Figure 21-17 shows the outbound window attributes register 0 (PEXOWAR0).

Offset 0xC10 Access: Mixed

	0	1	2	3	4	5–7	8–10	11	12–15	16–19	20–25	26–31
R W	EN	—		ROE	NS	—	TC	—	RTT	WTT	—	OWS
Reset	1	0	0	0	0	0 0 0	0 0 0	0	0 1 0 0	0 1 0 0	0 0 0 0 0 0	1 0 0 0 1 1

Figure 21-17. PCI Express Outbound Window Attributes Register 0 (PEXOWAR0)

Figure 21-18 shows the PCI Express outbound window attributes registers 1, 2, 4 (PEXOWAR*n*).

Offset Window 1: 0xC30 Access: Read/Write
Window 2: 0xC50
Window 4: 0xC90

	0	1	2	3	4	5–7	8–10	11	12–15	16–19	20–25	26–31
R W	EN	—	—	ROE	NS	—	TC	—	RTT	WTT	—	OWS
Reset	0	0	0	0	0	0 0 0	0 0 0	0	0 1 0 0	0 1 0 0	0 0 0 0 0 0	1 0 0 0 1 1

Figure 21-18. PCI Express Outbound Window Attributes Registers 1, 2, 4 (PEXOWAR*n*)

Figure 21-19 shows the PCI Express outbound window attributes registers 3 (PEXOWAR3).

Offset Window 3: 0xC70 Access: Read/Write

	0	1	2	3	4	5 7	8 10	11	12 15	16 19	20 25	26 31
R	EN	—	—	ROE	NS	—	TC	—	RTT	WTT	—	OWS
W												
Reset[1]	All zeros											

Figure 21-19. PCI Express Outbound Window Attributes Register 3 (PEXOWAR3)

1 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOWAR3 is 0x8004_400F. If the device is not configured to use the alternate boot vector (cfg_boot_vec = 1), the reset value for PCI controller 1 PEXOWAR3 is all zeros.

Table 21-18 describes the fields of the PCI Express outbound window attributes registers.

Table 21-18. PEXOWAR*n* Field Descriptions

Bits	Name	Description
0	EN	Enable. This bit enables this address translation window. For the default window, this bit is read-only and always hardwired to 1. 0 Disable outbound translation window 1 Enable outbound translation window
1–2	—	Reserved
3	ROE	Relaxed ordering enable. This bit when set and the PCI Express device control register[Enable Relaxed] bit is set enables the Relaxed Ordering bit for the packet. This bit only applies to memory transactions. 0 Default ordering 1 Relaxed ordering
4	NS	No snoop enable. This bit when set and the PCI Express device control register[Enable No Snoop] bit is set enables the no snoop bit for the packet. This bit only applies to memory transactions. 0 Snoopable 1 No snoop
5–7	—	Reserved
8–10	TC	Traffic class. This field indicates the traffic class of the outbound packet. This field only applies to memory transaction. All other transaction types should set the TC field to 0. 000 TC0 001 TC1 010 TC2 011 TC3 100 TC4 101 TC5 110 TC6 111 TC7 **Note:** Traffic class settings are passed through to the PCI Express link, but no specific actions are taken in the device based on traffic class.
11	—	Reserved

Table 21-18. PEXOWAR*n* Field Descriptions (continued)

Bits	Name	Description
12–15	RTT	Read transaction type. Read transaction type to run on the PCI Express link 0000 Reserved 0000 Reserved 0010 Configuration read. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary. 0100 Memory read ... Reserved 1000 IO read. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary. ... Reserved 1111 Reserved
16–19	WTT	Write transaction type. Write transaction type to run on the PCI Express link. 0000 Reserved 0001 Reserved 0010 Configuration write. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary. Note that inbound write transactions on one PCI express port must not translate to outbound configuration write transactions on another PCI Express port. 0100 Memory write 0101 Message write. Only support 4-byte size access on a 4-byte address boundary. ... Reserved 1000 IO Write. Supported only when in RC mode and size of less than or equal to 4 bytes and not crossing 4-byte address boundary. Note that inbound write transactions on one PCI express port must not translate to outbound I/O write transactions on another PCI Express port. ... Reserved 1111 Reserved
20–25	—	Reserved
26–31	OWS	Outbound window size. Outbound translation window size N which is the encoded $2^{(N + 1)}$-byte window size. The smallest window size is 4 Kbytes. Note that for the default window (window 0), the outbound window size may be programmed less than the 64-Gbyte maximum. However, accesses that miss all other windows and hit outside the default window is aliased to the default window. 000000Reserved ... 0010114-Kbyte window size 0011008-Kbyte window size ... 0111114-Gbyte window size 1000008-Gbyte window size 10000116-Gbyte window size 10001032-Gbyte window size 10001164-Gbyte window size 100100Reserved ... 111111Reserved

21.3.5.2 PCI Express Inbound ATMU Registers

There are differences between RC and EP implementations of inbound ATMU registers as described in the following sections.

21.3.5.2.1 EP Inbound ATMU Implementation

All base address registers (BARs) reside in the PCI Express type 0 configuration header space which is accessible through the PEX_CONFIG_ADDR/PEX_CONFIG_DATA mechanism. Note that host software must program these BAR using configuration type 0 cycles. There are 4 inbound BARs.

- Default inbound window BAR0 at configuration address 0x10 (32-bit). Also known as PEXCSRBAR. This is a fixed 1-Mbyte window used for inbound memory transactions that access memory-mapped registers. 1
- Inbound window BAR1 at configuration address 0x14 (32-bit)
- Inbound window BAR2 at configuration address 0x18–0x1c (64-bit)
- Inbound window BAR3 at configuration address 0x20–0x24 (64-bit)

The PCI Express controller does not implement a shadow of the inbound BARs in the memory-mapped register set. However, when there is a hit to the BAR(s), the PCI Express controller uses the corresponding translation and attribute registers in the memory-mapped register set for the translation. If the transaction hits multiple BARs, then the lowest-numbered BAR is used.

21.3.5.2.2 RC Inbound ATMU Implementation

In RC mode, the PEXIWBAR[1–3] registers reside outside of the type 1 header; PEXIWBAR0 is the only inbound BAR that resides in the Type 1 header (at offset 0x10).

If the transaction hits any window, the translation is performed and then the transaction is sent to memory. If there is no hit to any one of the BARs, then an UR completion is returned for non-posted transactions. All posted transactions with no BAR hit are ignored.

Figure 21-20 shows the inbound transaction flow in RC mode.

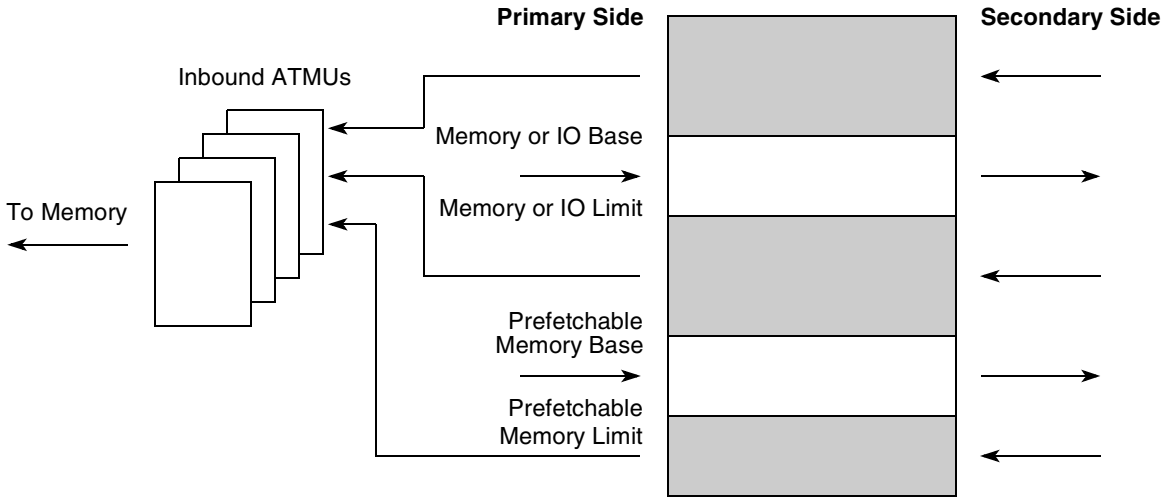


Figure 21-20. RC Inbound Transaction Flow

21.3.5.2.3 PCI Express Inbound Translation Address Registers (PEXITAR*n*)

The PCI Express inbound translation address registers, shown in Figure 21-21, contain the translated internal platform address to be used. Note that PEXITAR0 does not exist in the memory-mapped space; it is a fixed translation to the internal configuration (CCSRBAR) space.

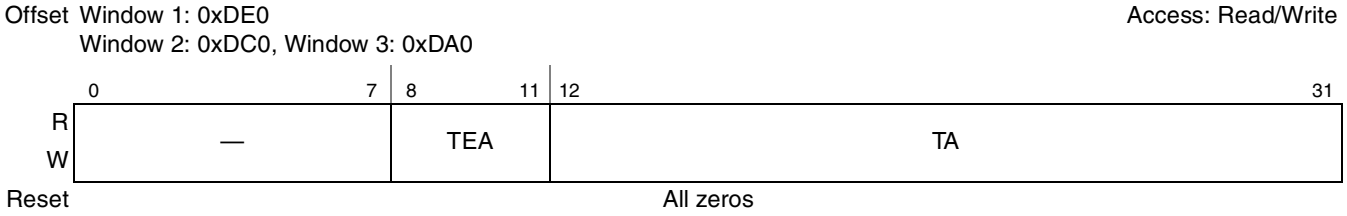


Figure 21-21. PCI Express Inbound Translation Address Registers (PEXITAR*n*)

Table 21-19 describes the fields of the PCI Express inbound translation address registers.

Table 21-19. PCI Express Inbound Translation Address Registers Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8–11	TEA	Translation extended address. Target address which indicates the starting point of the inbound translated address. The translation address must be aligned based on the size field. Corresponds to internal platform address bits [0:3] where bit 0 is the msb of the internal platform address.
12–31	TA	Translation address. Target address which indicates the starting point of the inbound translated address. The translation address must be aligned based on the size field. This corresponds to internal platform address bits [4:23].

21.3.5.2.4 PCI Express Inbound Window Base Address Registers (PEXIWBAR*n*)

The PCI Express inbound window base address registers, shown in Figure 21-22, select the base address for the windows which are translated to an alternate target address space. In root complex (RC) mode, addresses for inbound transactions are compared to these windows. In RC mode, PEXIWBAR0 is located in the PCI Express type 1 configuration header space and PEXIWBAR[1–3] registers are implemented as described in this section. In endpoint (EP) mode, these registers are not implemented in the memory-mapped space. Reading these registers in EP mode returns all zeros and writing to these offsets has no consequences. All base address registers in EP are located in the PCI Express type 0 configuration header space. Note that PEXIWBAR1 only supports 32-bit PCI Express address space.

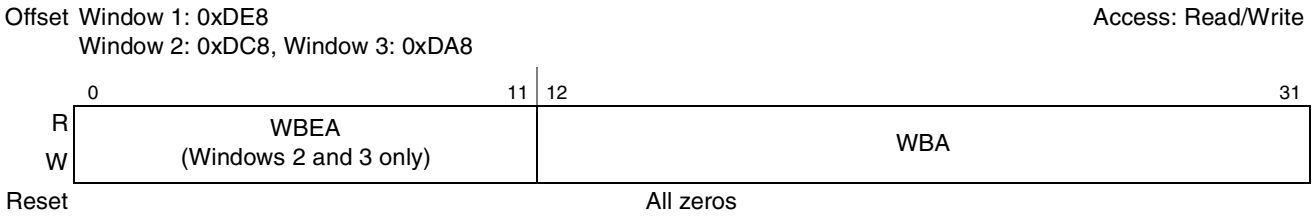


Figure 21-22. PCI Express Inbound Window Base Address Registers (PEXIWBAR*n*)

Table 21-20 describes the fields of the PCI Express inbound window base address registers.

Table 21-20. PCI Express Inbound Window Base Address Register Field Descriptions

Bits	Name	Description
0–11	WBEA	Window base extended address. This field corresponds to PCI Express address bits [43:32]. Note that the extended address is supported for windows 2 and 3 only; for PEXIWBAR1, these bits are reserved and must be 0.
12–31	WBA	Window base address. Source address which is the starting point for the inbound translation window. The window must be aligned based on the size selected in the window size bits. This corresponds to PCI Express address bits [31:12].

21.3.5.2.5 PCI Express Inbound Window Base Extended Address Registers (PEXIWBEAR*n*)

The PCI Express inbound window base extended address registers, shown in Figure 21-23, contain the most-significant bits of a 64 bit base address.

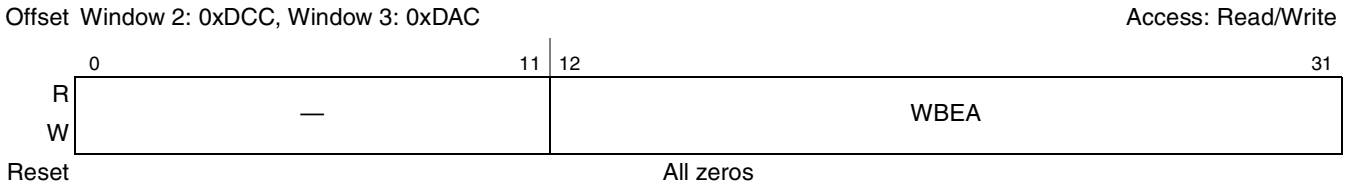


Figure 21-23. PCI Express Inbound Window Base Extended Address Registers (PEXIWBEAR*n*)

Table 21-21 describes the fields of the PCI Express inbound window base extended address registers.

Table 21-21. PCI Express Inbound Window Base Extended Address Register Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–31	WBEA	Window base extended address. This field corresponds to PCI Express address bits [63:44]

21.3.5.2.6 PCI Express Inbound Window Attributes Registers (PEXIWAR*n*)

The PCI Express inbound window attributes registers, shown in Figure 21-24, define the window sizes to translate and other attributes for the translations. 64 Gbytes is the largest window size allowed.

Offset Window 1: 0xDF0 Access: Read/Write
Window 2: 0xDD0, Window 3: 0xDB0

	0	1	2	3–7	8–11	12–15	16–19	20–25	26–31
R/W	EN	—	PF	—	TRGT	RTT	WTT	—	IWS
Reset	0	0	1	0 0 0 0 0	1 1 1 1	0 1 0 0	0 1 0 0	0 0 0 0 0 0	1 0 0 0 1 1

Figure 21-24. PCI Express Inbound Window Attributes Registers (PEXIWAR*n*)

Table 21-22 describes the fields of the PCI Express inbound window attributes registers.

Table 21-22. PCI Express Inbound Window Attributes Registers Field Descriptions

Bits	Name	Description
0	EN	Enable. This bit controls the enabling/disabling of the translation window. 0 Disable inbound window translation 1 Enable inbound window translation
1	—	Reserved
2	PF	Prefetchable. This bit indicates that the address space is prefetchable. This bit corresponds to the prefetchable bit in the BAR in the PCI Express type 0 header. This bit drives the BAR's prefetchable bit in EP mode. 0 Not prefetchable 1 Prefetchable
3–7	—	Reserved

Table 21-22. PCI Express Inbound Window Attributes Registers Field Descriptions (continued)

Bits	Name	Description
8–11	TRGT	Target interface. PEXIWAR*n*[TRGT] must not be set to target the PCI Express interface itself. That is, PEXIWAR*n* for PCI Express controller 1 should never target PCI Express 1. For PCI Express controller 2, the only supported target is the local address space. Note that if this field is set to anything other than local address space, the attributes for the transaction must be assigned in a corresponding outbound window at the target. 0000 PCI on OCN1; reserved on OCN2 0001 Reserved 0010 Reserved 0011 Reserved 0100 Reserved 0101 Reserved 0110 Reserved 0111 Reserved 1000 Reserved 1001 Reserved 1010 Reserved 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 Local address space (DDR or eLBC)
12–15	RTT	Read transaction type. Read transaction type to send to target interface. If the transaction is not going to local memory space 0000 Reserved ... 0100 Read 0101 Reserved 0110 Reserved 0111 Reserved 1000 Reserved ... 1111 Reserved If the transaction is going to local memory space 0000 Reserved ... 0100 Read, do not snoop local processor 0101 Read, snoop local processor 0110 Reserved 0111 Reserved 1000 Reserved ... 1111 Reserved

Table 21-22. PCI Express Inbound Window Attributes Registers Field Descriptions (continued)

Bits	Name	Description
16–19	WTT	Write transaction type. Write transaction type to send to target interface. If the transaction is not going to local memory space 0000 Reserved ... 0100 Write 0101 Reserved 0110 Reserved 0111 Reserved 1000 Reserved ... 1111 Reserved If the transaction is going to local memory space 0000 Reserved ... 0100 Write, do not snoop local processor 0101 Write, snoop local processor 0110 Reserved 0111 Reserved 1000 Reserved ... 1111 Reserved
20–25	—	Reserved
26–31	IWS	Inbound window size. Inbound translation window size N which is the encoded $2^{(N + 1)}$-bytes window size. The smallest window size is 4 Kbytes. For EP mode, this field directly controls the size of the BARs. 000000 Reserved ... 001010 Reserved 001011 4-Kbyte window size 001100 8-Kbyte window size ... 011111 4-Gbyte window size 100000 8-Gbyte window size 100001 16-Gbyte window size 100010 32-Gbyte window size 100011 64-Gbyte window size 100100 Reserved ... 111111 Reserved

21.3.6 1PCI Express Error Management Registers

21.3.6.1 PCI Express Error Detect Register (PEX_ERR_DR)

The PCI Express error detect register, shown in Figure 21-25, contains error status bits that are detected by hardware.The detected error bits are write-1-to-clear type registers. Reading from these registers 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 bit 6 and not affect any

other bits in the register, the value 0b0200_0000 is written to the register. When an error is detected the appropriate error bit is set. Subsequent errors sets the appropriate error bits in the error detection registers, but only the first error for a particular unit have any relevant information captured. The interrupt enable bits are used to allow or block the error reporting to the interrupt mechanism while the disable bits are used to prevent or allow the setting of the status bits.

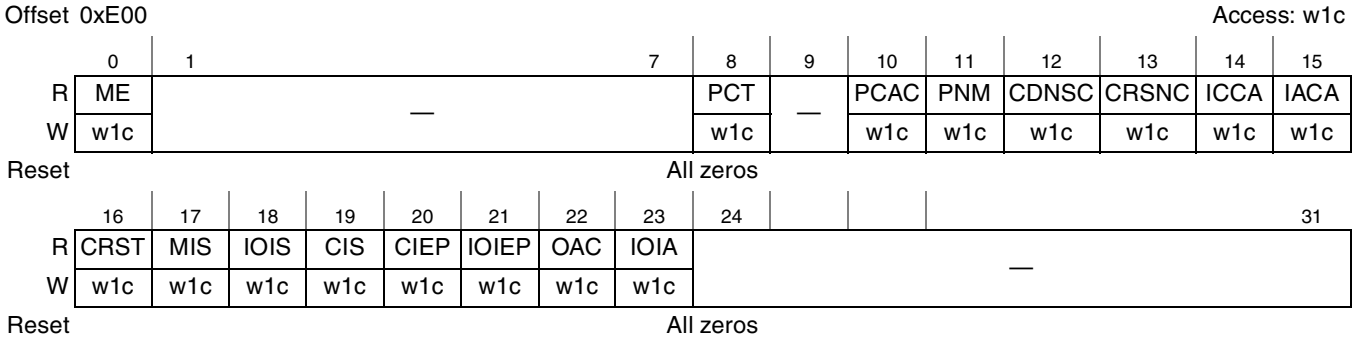


Figure 21-25. PCI Express Error Detect Register (PEX_ERR_DR)

Table 21-23 describes the fields of the PCI Express error detect register.

Table 21-23. PCI Express Error Detect Register Field Descriptions

Bits	Name	Description
0	ME	Multiple errors. Detecting multiple errors of the same type. An error is considered as multiple error when its detect bit is set and the same error is occurring again. Note that this bit does not track the ordering of when the error occurs. 1 Multiple errors were detected. 0 Multiple errors were not detected.
1–7	—	Reserved
8	PCT	PCI Express completion time-out. A completion time-out condition was detected for a non-posted, outbound PCI Express transaction. An error response is sent back to the requestor. Note that a completion timeout counter only starts when the non-posted request was able to send to the link partner. 1 A completion time-out on the PCI Express link was detected. Note that a completion timeout error is a fatal error. If a completion timeout error is detected, the system has become unstable. Hot reset is recommended to restore stability of the system. 0 No completion time-out on the PCI Express link detected.
9	—	Reserved
10	PCAC	PCI Express completer abort (CA) completion. A completion with CA status was received. 1 A completion with CA status was detected. 0 No completion with CA status detected.
11	PNM	PCI Express no map. An inbound transaction that is not mapped to any inbound windows was detected. In RC mode, a posted transaction will be dropped silently and this bit will be set. A nonposted transaction will return a completion without data (Cpl) packet with a UR completion status to the requester and this bit is set. For EP mode, a Cpl packet with a UR completion status is sent back to the requester but does not set this bit.. 1 A no-map transaction was detected in RC mode. 0 No no-map transaction detected.

Table 21-23. PCI Express Error Detect Register Field Descriptions (continued)

Bits	Name	Description
12	CDNSC	Completion with data not successful. A completion with data packet was received with a non successful status (that is, UR, CA or CRS status). 1 Completion with data non successful packet was detected. 0 No completion with data non successful packet detected.
13	CRSNC	CRS non configuration. A completion was detected for a non configuration cycle and with CRS status. See Section 21.3.7.1.1, "PCI Express Configuration Access Register Mechanism," for more information. 1 CRS non configuration packet was detected. 0 No CRS non configuration packet detected.
14	ICCA	Invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA configuration access. Access to an illegal configuration space from PEX_CONFIG_ADDR/PEX_CONFIG_DATA was detected. 1 Invalid CONFIG_ADDR/PEX_CONFIG_DATA access detected 0 No invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access detected
15	IACA	Invalid ATMU configuration access. Access to an illegal configuration space from an ATMU window was detected. 1 Invalid ATMU configuration access was detected 0 No invalid ATMU configuration access detected
16	CRST	CRS thresholded. An outbound configuration transaction was retried and thresholded due to a CRS completion status. An error response is sent back to the requestor. See Section 21.3.2.4, "PCI Express Configuration Retry Timeout Register (PEX_CONF_RTY_TOR)," for more information. 1 A CRS threshold condition was detected for an outbound configuration transaction 0 No CRS threshold condition detected
17	MIS	Message invalid size. An outbound message transaction that is greater than 4 bytes or crosses a 4-byte boundary was detected. See Section 21.4.1.9.1, "Outbound ATMU Message Generation," for more information. 1 An invalid size outbound message transaction was detected 0 No invalid size outbound message transaction detected
18	IOIS	I/O invalid size. An outbound I/O transaction that is greater than 4 bytes or crosses a 4-byte boundary was detected. 1 an invalid size outbound I/O transaction was detected 0 no invalid size outbound I/O transaction detected
19	CIS	Configuration invalid size. An outbound configuration transaction that is greater than 4 bytes or crosses a 4-byte boundary was detected. 1 An invalid size outbound configuration transaction was detected 0 No invalid size outbound configuration transaction detected
20	CIEP	Configuration invalid EP. An outbound ATMU configuration transaction request was seen when in EP mode. 1 An outbound configuration transaction while in EP was detected 0 No outbound configuration transaction in EP detected
21	IOIEP	I/O invalid EP. An outbound I/O transaction request was seen when in EP mode. 1 An outbound I/O transaction while in EP was detected 0 No outbound I/O transaction in EP detected
22	OAC	Outbound ATMU crossing. An outbound crossing ATMU transaction was detected. 1 An outbound transaction that hits in one window and crosses overing it was detected 0 No outbound ATMU crossing condition detected

Table 21-23. PCI Express Error Detect Register Field Descriptions (continued)

Bits	Name	Description
23	IOIA	I/O invalid address. An outbound I/O transaction with a translated address of greater than 4 Gbytes was detected. 1 A greater than 4-Gbyte I/O address was detected 0 No greater than 4-Gbyte I/O address detected
24–31	—	Reserved

21.3.6.2 PCI Express Error Interrupt Enable Register (PEX_ERR_EN)

The PCI Express error interrupt enable register, shown in Figure 21-26, allows interrupts to be generated when the corresponding PCI Express error detect register bits are set.

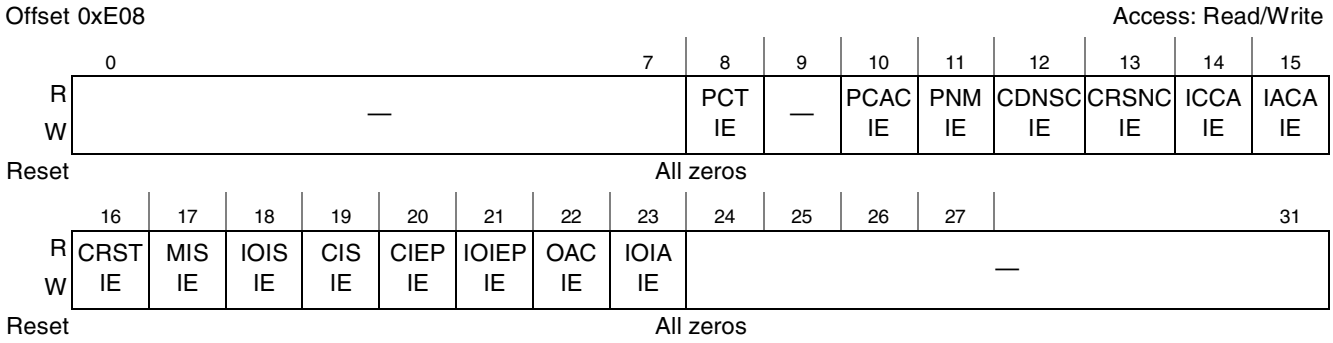


Figure 21-26. PCI Express Error Interrupt Enable Register (PEX_ERR_EN)

Table 21-24 describes the fields of the PCI Express error interrupt enable register.

Table 21-24. PCI Express Error Interrupt Enable Register Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8	PCTIE	PCI Express completion time-out interrupt enable. When set and PEX_ERR_DR[PCT]=1 generates an interrupt. 1 Enable PCI Express completion time-out interrupt generation 0 Disable PCI Express completion time-out interrupt generation
9	—	Reserved
10	PCACIE	PCI Express CA completion interrupt enable. When set and PEX_ERR_DR[PCAC]=1 generates an interrupt. 1 Enable completion with CA status interrupt generation 0 Disable completion with CA status interrupt generation
11	PNMIE	PCI Express no map interrupt enable. When set and PEX_ERR_DR[PNM]=1 generates an interrupt. 1 Enable no map PCI Express packet interrupt generation 0 Disable no map PCI Express packet interrupt generation
12	CDNSCIE	Completion with data not successful interrupt enable. When this bit is set and PEX_ERR_DR[CDNSC] = 1 generates an interrupt. 1 Enable completion with data non successful interrupt generation 0 Disable completion with data non successful interrupt generation

Table 21-24. PCI Express Error Interrupt Enable Register Field Descriptions (continued)

Bits	Name	Description
13	CRSNCIE	CRS non configuration interrupt enable. When this bit is set and PEX_ERR_DR[CRSNC] = 1 generates an interrupt. 1 Enable CRS non configuration interrupt generation 0 Disable CRS non configuration interrupt generation
14	ICCAIE	Invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA configuration access interrupt enable. When set and PEX_ERR_DR[ICCA]=1 generates an interrupt. 1 Enable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access interrupt generation 0 Disable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access interrupt generation.
15	IACAIE	Invalid ATMU configuration access. When set and PEX_ERR_DR[IACA]=1 generates an interrupt. 1 Enable invalid ATMU configuration access interrupt generation 0 Disable invalid ATMU configuration access interrupt generation
16	CRSTIE	CRS thresholded interrupt enable. When set and PEX_ERR_DR[CRST]=1 generates an interrupt. 1 Enable CRS threshold interrupt generation 0 Disable CRS threshold interrupt generation
17	MISIE	Message invalid size interrupt enable. When set and PEX_ERR_DR[MIS]=1 generates an interrupt. 1 Enable invalid outbound message size interrupt generation 0 Disable invalid outbound message size interrupt generation
18	IOISIE	I/O invalid size interrupt enable. When set and PEX_ERR_DR[IOIS]=1 generates an interrupt. 1 Enable invalid outbound I/O size interrupt generation 0 Disable invalid outbound I/O size interrupt generation
19	CISIE	Configuration invalid size interrupt enable. When set and PEX_ERR_DR[CIS]=1 generates an interrupt. 1 Enable invalid outbound configuration size interrupt generation 0 Disable invalid outbound configuration size interrupt generation
20	CIEPIE	Configuration invalid EP interrupt enable. When set and PEX_ERR_DR[CIEP]=1 generates an interrupt. 1 Enable outbound configuration transaction while in EP mode interrupt generation 0 Disable outbound configuration transaction in EP mode interrupt generation
21	IOIEPIE	I/O invalid EP interrupt enable. When set and PEX_ERR_DR[IOIEP]=1 generates an interrupt. 1 Enable outbound I/O transaction EP mode interrupt generation 0 Disable outbound I/O transaction EP mode interrupt generation
22	OACIE	Outbound ATMU crossing interrupt enable. When set and PEX_ERR_DR[OAC]=1 generates an interrupt. 1 Enable outbound crossing ATMU interrupt generation 0 Disable outbound crossing ATMU interrupt generation
23	IOIAIE	I/O address invalid enable. When set and PEX_ERR_DR[IOIA]=1 generates an interrupt. 1 Enable greater than 4G I/O address interrupt generation 0 Disable greater than 4G I/O address interrupt generation
24–31	—	Reserved

21.3.6.3 PCI Express Error Disable Register (PEX_ERR_DISR)

The PCI Express error disable register, shown in Figure 21-27, controls the setting of the PCI Express error detect register's bits.

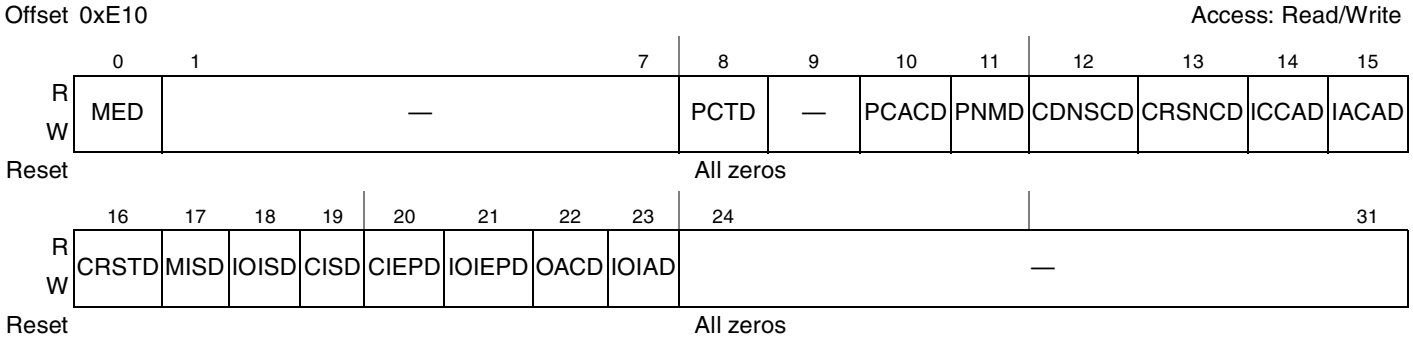


Figure 21-27. PCI Express Error Disable Register (PEX_ERR_DISR)

Table 21-25 describes the fields of the PCI Express error disable register.

Table 21-25. PCI Express Error Disable Register Field Descriptions

Bits	Name	Description
0	MED	Multiple errors disable. When set disables the setting of PEX_ERR_DR[ME] bit. 1 Disable multiple errors detection 0 Enable multiple errors detection
1–7	—	Reserved
8	PCTD	PCI Express completion time-out disable. When set disables the setting of PEX_ERR_DR[PCT] bit. 1 Disable PCI Express completion time-out detection 0 Enable PCI Express completion time-out detection
9	—	Reserved
10	PCACD	PCI Express CA completion disable. When set disables the setting of PEX_ERR_DR[PCAC] bit. 1 Disable completion with CA status detection 0 Enable completion with CA status detection
11	PNMD	PCI Express no map disable. When set disables the setting of PEX_ERR_DR[PNM] bit. 1 Disable no map PCI Express packet detection 0 Enable no map PCI Express packet detection
12	CDNSCD	Completion with data not successful disable. When set disables the setting of PEX_ERR_DR[CDNSC] bit. 1 Disable completion with data not successful detection 0 Enable completion with data not successful detection
13	CRSNCD	CRS non configuration disable. When set disables the setting of PEX_ERR_DR[CRSNC] bit. 1 Disable CRS non configuration detection 0 Enable CRS non configuration detection
14	ICCAD	Invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA configuration access disable. When set disables the setting of PEX_ERR_DR[ICCA] bit. 1 Disable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access detection 0 Enable invalid PEX_CONFIG_ADDR/PEX_CONFIG_DATA access detection

Table 21-25. PCI Express Error Disable Register Field Descriptions (continued)

Bits	Name	Description
15	IACAD	Invalid ATMU configuration access. When set disables the setting of PEX_ERR_DR[IACA] bit. 1 Disable invalid ATMU configuration access detection 0 Enable invalid ATMU configuration access detection
16	CRSTD	CRS thresholded disable. When set disables the setting of PEX_ERR_DR[CRST] bit. 1 Disable CRS threshold detection 0 Enable CRS threshold detection
17	MISD	Message invalid size disable. When set disables the setting of PEX_ERR_DR[MIS] bit. 1 Disable invalid outbound message size detection 0 Enable invalid outbound message size detection
18	IOISD	I/O invalid size disable. When set disables the setting of PEX_ERR_DR[IOIS] bit. 1 Disable invalid outbound I/O size detection 0 Enable invalid outbound I/O size detection
19	CISD	Configuration invalid size disable. When set disables the setting of PEX_ERR_DR[CIS] bit. 1 Disable invalid outbound configuration size detection 0 Enable invalid outbound configuration size detection
20	CIEPD	Configuration invalid EP disable. When set disables the setting of PEX_ERR_DR[CIEP] bit. 1 Disable outbound configuration transaction EP mode detection 0 Enable outbound configuration transaction EP mode detection
21	IOIEPD	I/O invalid EP disable. When set disables the setting of PEX_ERR_DR[IOEP] bit. 1 Disable outbound I/O transaction EP mode detection 0 Enable outbound I/O transaction EP mode detection
22	OACD	Outbound ATMU crossing disable. When set disables the setting of PEX_ERR_DR[OAC] bit. 1 Disable outbound crossing ATMU detection 0 Enable outbound crossing ATMU detection
23	IOIAD	I/O invalid address disable. When set disables the setting of PEX_ERR_DR[IOIA] bit. 1 Disable greater than 4G I/O address detection 0 Enable greater than 4G I/O address detection
24–31	—	Reserved

21.3.6.4 PCI Express Error Capture Status Register (PEX_ERR_CAP_STAT)

The PCI Express error capture status register, shown in Figure 21-28, allows vital error information to be captured when an error occurs. Note that no further error capturing is performed until the ECV bit is cleared.

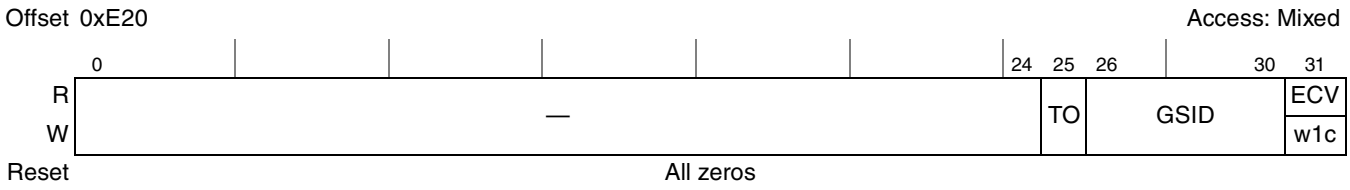


Figure 21-28. PCI Express Error Capture Status Register (PEX_ERR_CAP_STAT)

Table 21-26 describes the fields of the PCI Express error capture status register.

Table 21-26. PCI Express Error Capture Status Register Field Descriptions

Bits	Name	Description
0–24	—	Reserved
25	TO	Transaction originator. This field Indicates whether the originator of the transaction is from PEX_CONFIG_ADDR/PEX_CONFIG_DATA. 1 Transaction originated from PEX_CONFIG_ADDR/PEX_CONFIG_DATA. 0 Transaction not originated from PEX_CONFIG_ADDR/PEX_CONFIG_DATA.
26–30	GSID	Global source ID. This field indicates the internal platform global source ID that the error transaction originates. This field only applies to non PEX_CONFIG_ADDR/PEX_CONFIG_DATA transactions. 00000 PCI 00010 PCI Express 2 (x8) 00011 PCI Express 1 (x4) 01001 DIU 01010 Boot sequencer 01111 DDR memory controller 10000 e600 instruction fetch 10001 e600 data access 10101 DMA1 10110 DMA2 All other settings reserved.
31	ECV	Error capture valid. This bit indicates that the capture registers 0–3 contain valid info. This bit when set indicates that the captured registers contain valid capturing information. No new capturing is done unless this bit is cleared by writing a 1 to it.

21.3.6.5 PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R0 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

21.3.6.5.1 PEX_ERR_CAP_R0—Outbound Case

PEX_ERR_CAP_R0 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02 for controller 1 or 0h03 for controller 2) and the error is due to timeout condition or PEX_CONFIG_ADDR/PEX_CONFIG_DATA access, is shown in Figure 21-29.

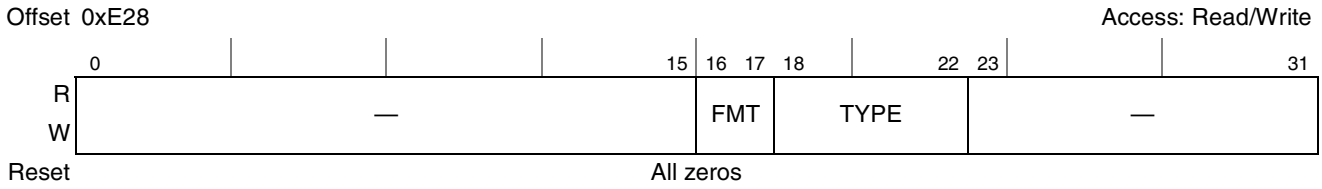


Figure 21-29. PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0) Internal Source, Outbound Transaction

Table 21-27 describes the fields of the PCI Express error capture register 0 for the case when the error is caused by an outbound transaction from an internal source.

Table 21-27. PCI Express Error Capture Register 0 Field Descriptions Internal Source, Outbound Transaction

Bits	Name	Description
0–15	—	Reserved
16-17	FMT	PCI Express format. This field indicates the PCI Express packet format. See PCI Express Spec 1.0a for more information on 3 or 4 DW (4-byte) header format.
18–22	TYPE	PCI Express type. This field indicates the PCI express packet type. See PCI Express Spec 1.0a .for more information on 3 or 4 DW (4-byte) header format.
23–31	—	Reserved

21.3.6.5.2 PEX_ERR_CAP_R0—Inbound Case

PEX_ERR_CAP_R0 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02), is shown in Figure 21-30. Inbound transactions that result in PEX_ERR_DR[PNM] or PEX_ERR_DR[PCAC] or PEX_ERR_DR[CDNSC] or PEX_ERR_DR[CRSNC] being set will result in PEX_ERR_CAP_STAT[GSID] = 0h02.

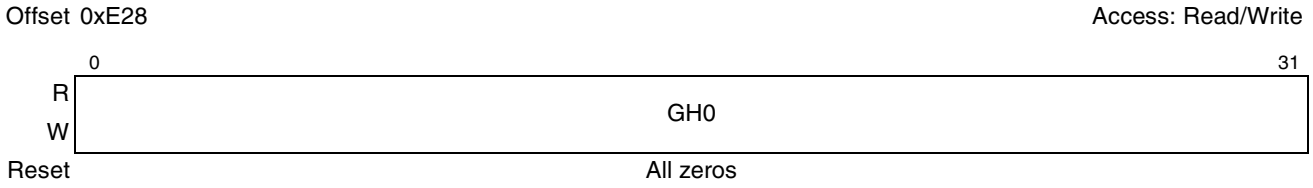


Figure 21-30. PCI Express Error Capture Register 0 (PEX_ERR_CAP_R0) External Source, Inbound Transaction

Table 21-28 describes the fields of PEX_ERR_CAP_R0 for the case when the error is caused by an inbound transaction from an external source.

Table 21-28. PCI Express Error Capture Register 0 Field Descriptions External Source, Inbound Transaction

Bits	Name	Description
0–31	GH0	PCI Express first DW (4-byte) header. This field contains the first DW (4-byte) of the captured PCI Express packet header. 27–31TYPE 25–26FMT 20–24Reserved 17–19TC 16 Reserved 14–15LENGTH[9:8] 12–13Reserved 10–11ATTR 9 EP 8 TD 0–7LENGTH[7:0]

21.3.6.6 PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R1 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

21.3.6.6.1 PEX_ERR_CAP_R1—Outbound Case

PEX_ERR_CAP_R1 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02 for controller 1 or 0h03 for controller 2), is shown in Figure 21-31.

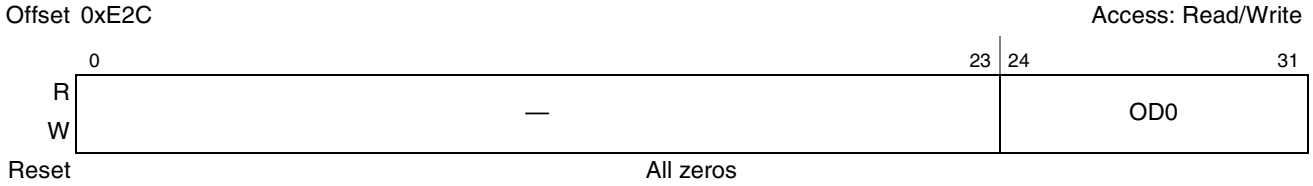


Figure 21-31. PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1) Internal Source, Outbound Transaction

Table 21-29 describes the fields of PEX_ERR_CAP_R1 for the case when the error is caused by an outbound transaction from an internal source.

Table 21-29. PCI Express Error Capture Register 1 Field Descriptions Internal Source, Outbound Transaction

Bits	Name	Description
0–23	—	Reserved
24–31	OD0	Internal platform transaction information. Reserved for factory debug.

21.3.6.6.2 PEX_ERR_CAP_R1—Inbound Case

PEX_ERR_CAP_R1 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02 or 0h03 for controller 2), is shown in Figure 21-32.

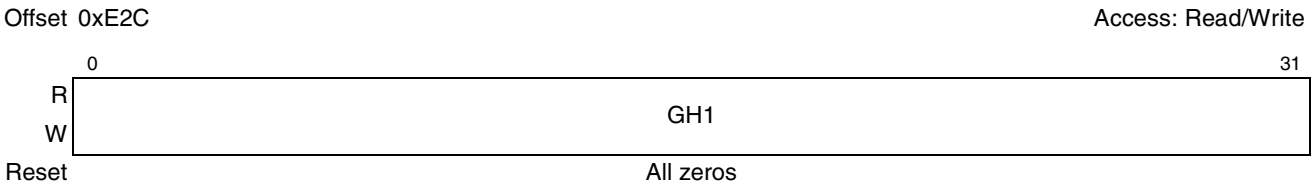


Figure 21-32. PCI Express Error Capture Register 1 (PEX_ERR_CAP_R1) External Source, Inbound Transaction

Table 21-30 describes the fields of PEX_ERR_CAP_R1 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 21-28) indicate the error was caused by an inbound completion transaction.

Table 21-30. PCI Express Error Capture Register 1 Field Descriptions External Source, Inbound Completion Transaction

Bits	Name	Description
0–31	GH1	PEX second DW (4-byte) header. This field contains the second DW (4-byte) of the captured PCI Express packet header. 24–31 Comp ID[15:8] 16–23 Comp ID[7:0] 12–15 Byte Count[11:8] 11 BCM 8–10 Comp Status 0–7 Byte Count[7:0]

Table 21-31 describes the fields of PEX_ERR_CAP_R1 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 21-28) indicate the error was caused by an inbound memory request transaction.

Table 21-31. PCI Express Error Capture Register 1 Field Descriptions External Source, Inbound Memory Request Transaction

Bits	Name	Description
0–31	GH1	PEX second DW (4-byte) header. This field contains the second DW (4-byte) of the captured PCI Express packet header. 24–31 Requester ID[15:8] 16–23 Requester ID[7:0] 8–15 Tag[7:0] 4–7 First DW BE[3:0] 0–3 Last DW BE[3:0]

21.3.6.7 PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R2 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

21.3.6.7.1 PEX_ERR_CAP_R2—Outbound Case

PEX_ERR_CAP_R2 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02 for controller 1 or 0h03 for controller 2), is shown in Figure 21-33.

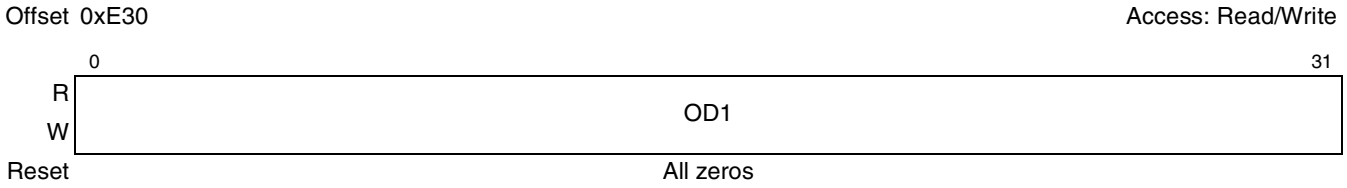


Figure 21-33. PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2) Internal Source, Outbound Transaction

Table 21-32 describes the fields of PEX_ERR_CAP_R2 for the case when the error is caused by an outbound transaction from an internal source.

Table 21-32. PCI Express Error Capture Register 2 Field Descriptions Internal Source, Outbound Transaction

Bit	Name	Description
0–31	OD1	Internal platform transaction information. Reserved for factory debug.

21.3.6.7.2 PEX_ERR_CAP_R2—Inbound Case

PEX_ERR_CAP_R2 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02), is shown in Figure 21-34.

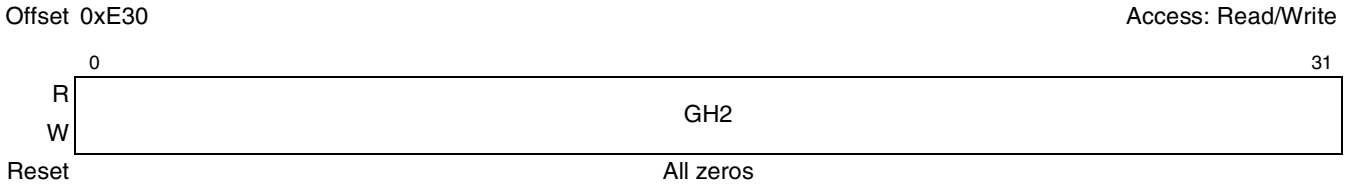


Figure 21-34. PCI Express Error Capture Register 2 (PEX_ERR_CAP_R2) External Source, Inbound Transaction

Table 21-33 describes the fields of PEX_ERR_CAP_R2 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 21-28) indicate the error was caused by an inbound completion transaction.

Table 21-33. PCI Express Error Capture Register 2 Field Descriptions External Source, Inbound Completion Transaction

Bits	Name	Description
0–31	GH2	PEX third DW (4-byte) header. This field contains the third DW (4-byte) of the captured PCI Express packet header. 24–31 Req ID[15:8] 16–23 Req ID[7:0] 8–15 Tag[7:0] 1–7 Lower Address[6:0] 0 Reserved

Table 21-34 describes the fields of PEX_ERR_CAP_R2 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 21-28) indicate the error was caused by an inbound memory request transaction. Note that PEX_ERR_CAP_R2 captures the 32-bit address for a 3 DW memory request header or the upper half of the 64-bit address for a 4 DW memory request header; the lower half of the 64-bit address for a 4 DW memory request header is captured in PEX_ERR_CAP_R3.

Table 21-34. PCI Express Error Capture Register 2 Field Descriptions External Source, Inbound Memory Request Transaction

Bits	Name	Description	
		3 DW Header	4 DW Header
0–31	GH2	PEX third DW (4-byte) header. This field contains the third DW (4-byte) of the captured PCI Express packet header.	
		24–31 Address[31:24] 16–23 Address[23:16] 8–15 Address[15:8] 6–7 Reserved 0–5 Address[7:2]	24–31 Address[63:56] 16–23 Address[55:48] 8–15 Address[47:40] 0–7 Address[39:32]

21.3.6.8 PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3)

Together with the other PCI Express error capture registers, PEX_ERR_CAP_R3 allows vital error information to be captured when an error occurs. Different error information is reported depending on whether the error source is from an outbound transaction from an internal source or from an inbound transaction from an external source; the source of the captured error is reflected in PEX_ERR_CAP_STAT[GSID]. Note that after the initial error is captured, no further capturing is performed until the PEX_ERR_CAP_STAT[ECV] bit is clear.

21.3.6.8.1 PEX_ERR_CAP_R3—Outbound Case

PEX_ERR_CAP_R3 for the case when the error is caused by an outbound transaction from an internal source (that is, PEX_ERR_CAP_STAT[GSID] ≠ 0h02 for controller 1 or 0h03 for controller 2), is shown in Figure 21-35.

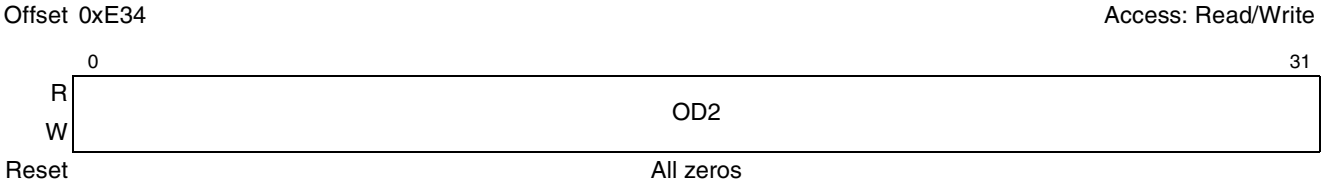


Figure 21-35. PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3) Internal Source, Outbound Transaction

Table 21-35 describes the fields of PEX_ERR_CAP_R3 for the case when the error is caused by an outbound transaction from an internal source.

Table 21-35. PCI Express Error Capture Register 3 Field Descriptions Internal Source, Outbound Transaction

Bits	Name	Description
0–31	OD2	Internal platform transaction information. Reserved for factory debug.

21.3.6.8.2 PEX_ERR_CAP_R3—Inbound Case

PEX_ERR_CAP_R3 for the case when the error is caused by an inbound transaction from an external source (that is, PEX_ERR_CAP_STAT[GSID] = 0h02), is shown in Figure 21-36.

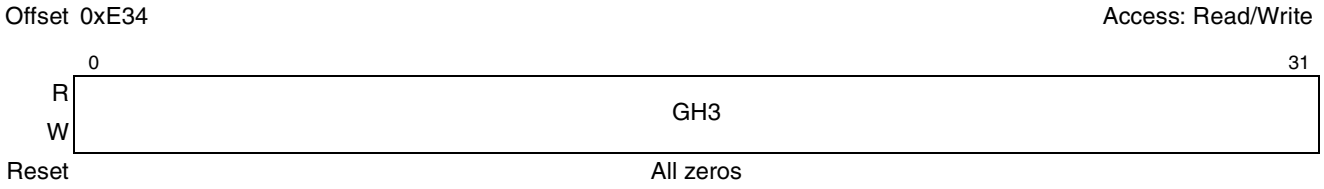


Figure 21-36. PCI Express Error Capture Register 3 (PEX_ERR_CAP_R3) External Source, Inbound Transaction

Table 21-36 describes the fields of PEX_ERR_CAP_R3 for the case when the FMT and TYPE subfields in PEX_ERR_CAP_R0 (see Table 21-28) indicate the error was caused by an inbound memory request transaction. Note that PEX_ERR_CAP_R3 captures the lower half of the 64-bit address for a 4 DW memory request header; the upper half of the 64-bit address for a 4 DW memory request header or the 32-bit address for a 3 DW memory request header is captured in PEX_ERR_CAP_R2.

Table 21-36. PEX Error Capture Register 3 Field Descriptions External Source, Inbound Memory Request Transaction

Bits	Name	Description
0–31	GH3	PEX fourth DW (4-byte) header. This field contains the fourth DW (4-byte) of the captured PCI Express packet header. 24–31 Address[31:24] 16–23 Address[23:16] 8–15 Address[15:8] 6–7 Reserved 0–5 Address[7:2]

21.3.7 PCI Express Configuration Space Access

There are two methods of accessing the PCI Express configuration header:

- PCI Express outbound ATMU window
- PCI Express configuration access registers (PEX_CONFIG_ADDR/PEX_CONFIG_DATA)

21.3.7.1 RC Configuration Register Access

To access internal configuration space, software must rely on the PCI Express configuration access register (PEX_CONFIG_ADDR/ PEX_CONFIG_DATA) mechanism. To access external configuration space, software can either use configuration access registers or the outbound ATMU mechanism. For the configuration access register method, a value must be written to the PEX_CONFIG_ADDR register that specifies the targeted PCI Express bus, the targeted device on that bus, the targeted function within the device, and the configuration register in that device that should be accessed. The PCI Express controller's bus number is obtained from the PCI Express configuration header (type 1). Then either a write or a read to the PEX_CONFIG_DATA register triggers the actual write or read cycle to the configuration space. Note that accesses to the little-endian PCI Express configuration space must be properly formatted. See Section 21.4.1.2.1, "Byte Order for Configuration Transactions," for more information.

Note that external configuration transactions should not be attempted until the link has successfully trained. Software can poll the LTSSM state status register (PEX_LTSSM_STAT) to check the status of link training before issuing external configuration requests.

21.3.7.1.1 PCI Express Configuration Access Register Mechanism

There are two types of configuration transactions (Type 0 and Type 1) needed to support hierarchical bridges.

- If the targeted bus number, and targeted device number equal to the PCI Express controller's bus number and device number, and the targeted function number is zero, then an internal PCI Express configuration cycle access is performed.
- If the targeted bus number does not equal the PCI Express controller's bus number, but does equal the secondary bus number (from the type 1 header) and the targeted device number is 0, then a Type 0 configuration transaction is sent to the PCI Express link.
- If the targeted bus number does not equal the PCI Express controller's bus number, and does not equal the secondary bus number (from the type 1 header), and the targeted bus number is less than or equal to the subordinate bus number (from the type 1 header), then a Type 1 configuration transaction is sent to the PCI Express link.
- If none of the above conditions occur, then the PCI Express controller returns all 1s for reads and ignores writes. In this case, PEX_ERR_DR[ICCA] is set.

21.3.7.1.2 Outbound ATMU Configuration Mechanism (RC-Only)

Software can also program one of the outbound ATMU windows to perform a configuration access. This is accomplished by programming the ReadTType or WriteTType field of the desired PEXOWAR to 0x2. Software must only issue 4-byte or less access to the ATMU configuration window and the access cannot cross a 4-byte boundary. The targeted bus number, targeted device number, targeted function number,

register, and targeted extended register number sent are decoded from the outbound translated PCI Express address.

- targeted bus number[7:0] = PCI Express address[27:20]
- targeted device number[4:0] = PCI Express address[19:15]
- targeted function number[2:0] = PCI Express address[14:12]
- targeted extended register number[3:0] = PCI Express address[11:8]
- targeted register number[5:0] = PCI Express address[7:2]

A Type 0 configuration cycle is sent to the link if the targeted bus number equals the secondary bus number (from the type 1 header) and targeted device number is 0. A Type 1 configuration cycle is sent to the link if targeted bus number does not equal primary bus and secondary bus numbers and it is less than or equal to the subordinate bus number (from the type 1 header). For all other cases, the PCI Express controller squashes the write and read results in a response with error returned. In this case, PEX_ERR_DR[IACA] is set.

Note that the PCI Express controller does not support access to its internal configuration registers using the outbound ATMU mechanism. That is, the outbound ATMU mechanism must not be used to program the internal registers.

21.3.7.2 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 all of the PCI Express configuration area except the PCI Express Controller Internal CSR registers in the extended PCI Express configuration space at offsets 0x400–0x6FF. The PCI Express Controller Internal CSR registers are not accessible by inbound PCI Express configuration transactions. Attempts to access these registers return all zeros.

While in EP mode, the PCI Express controller does not support generating configuration accesses as a master. All accesses to PEX_CONFIG_ADDR/PEX_CONFIG_DATA cause the device to access the internal configuration registers regardless of the targeted bus number or targeted device number programmed in the PEX_CONFIG_ADDR register. 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 get a response with an error.

21.3.8 PCI Compatible Configuration Headers

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 for all PCI Express devices. These common registers are shown in Figure 21-37.

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 21-37. 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. There are two header types applicable to PCI Express. Type 0 headers are typically used by endpoints; Type 1 headers are used by root complexes and switches/bridges.

21.3.8.1 Common PCI Compatible Configuration Header Registers

This section details the registers that are common to both type 0 and type 1 configuration headers.

21.3.8.1.1 PCI Express Vendor ID Register—Offset 0x00

The vendor ID register, shown in Figure 21-38, is used to identify the manufacturer of the device.

Offset 0x00 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 21-38. PCI Express Vendor ID Register

Table 21-37 describes the vendor ID register fields.

Table 21-37. PCI Express Vendor ID Register Field Description

Bits	Name	Description
15–0	Vendor ID	0x1957 (Freescale)

21.3.8.1.2 PCI Express Device ID Register—Offset 0x02

The device ID register, shown in Figure 21-39, is used to identify the device.

Offset 0x02 Access: Read only

	15	0
R	Device ID	
W		
Reset	Device-specific; see field description	

Figure 21-39. PCI Express Device ID Register

Table 21-38 describes the device ID register fields.

Table 21-38. PCI Express Device ID Register Field Description

Bits	Name	Description
15–0	Device ID	0x7018 MPC8610

21.3.8.1.3 PCI Express Command Register—Offset 0x04

The command register, shown in Figure 21-40, provides control over the ability to generate and respond to PCI Express cycles.

Offset 0x04 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 21-40. PCI Express Command Register

Table 21-39 describes the bits of the command register.

Table 21-39. PCI Express Command Register Field Descriptions

Bits	Name	Description
15–11	—	Reserved
10	Interrupt Disable	Controls the ability to generate INTx interrupt messages. 0 Enables INTx interrupt messages 1 Disables INTx interrupt messages Any INTx emulation interrupts already asserted by this device must be deasserted when this bit is set.
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 **Note:** 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 21.3.9.8, “PCI Express Device Control Register—0x54,” and the advance error reporting capability structure described in sections 21.3.10.1 through 21.3.10.12.
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. **Note:** 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 21.3.9.8, “PCI Express Device Control Register—0x54,” and the advance error reporting capability structure described in sections 21.3.10.1 through 21.3.10.12.
5–3	—	Reserved

Table 21-39. PCI Express Command Register Field Descriptions (continued)

Bits	Name	Description
2	Bus master	Indicates whether this PCI Express device is configured as a master. 0 Disables the ability to generate PCI Express accesses 1 Enables this PCI Express controller to behave as a PCI Express bus master EP mode: 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. RC mode: Clearing this bit disables the ability of the device to forward memory transactions upstream. This causes any inbound memory transaction to be treated as an unsupported request.
1	Memory space	Controls whether this PCI Express device (as a target) responds to memory accesses. 0 This PCI Express device does not respond to PCI Express memory space accesses. 1 This PCI Express device responds to PCI Express memory space accesses. EP mode: Clearing this bit prevents the device from accepting any memory transaction. RC mode: This bit is ignored. It does not affect outbound memory transaction
0	I/O space	I/O space. 0 This PCI Express device (as a target) does not respond to PCI Express I/O space accesses. 1 This PCI Express device (as a target) does respond to PCI Express I/O space accesses. EP mode: Clearing this bit prevents the device from accepting any IO transaction. Note that this bit is a don't care in EP mode since the device does not support IO transaction. RC mode: This bit is ignored. It does not affect outbound IO transaction.

21.3.8.1.4 PCI Express Status Register—Offset 0x06

The status register, shown in Figure 21-41, is used to record status information for PCI Express related events.

Offset 0x06 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	5	4	3	2	0
R	—		Capabilities list	Interrupt Status	—	
W						
Reset	0	0	0	1	0	0 0 0 0

Figure 21-41. PCI Express Status Register

The definition of each bit is given in Table 21-40.

Table 21-40. PCI Express Status Register Field Descriptions

Bits	Name	Description
15	Detected parity error[1]	Set whenever a device receives a poisoned TLP regardless of the state of bit 6 in the command register.
14	Signaled system error[1]	Set whenever 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 whenever a requestor receives a completion with unsupported request completion status.
12	Received target-abort[1]	Set whenever a device receives a completion with completer abort completion status.
11	Signaled target-abort[1]	Set whenever 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 for this bit to 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 21.3.9.8, "PCI Express Device Control Register—0x54," and the advance error reporting capability structure described in sections 21.3.10.1 through 21.3.10.12.

21.3.8.1.5 PCI Express Revision ID Register—Offset 0x08

The revision ID register, shown in Figure 21-42, is used to identify the revision of the device.

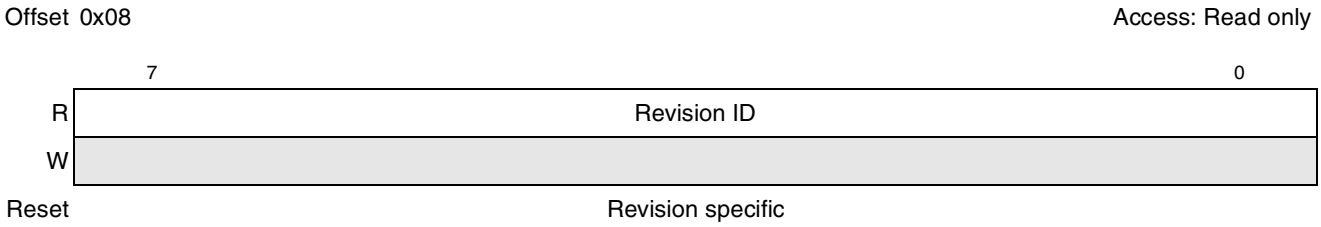


Figure 21-42. PCI Express Revision ID Register

Table 21-41 describes the revision ID register fields.

Table 21-41. PCI Express Revision ID Register Field Descriptions

Bits	Name	Description
7–0	Revision ID	Revision specific.

21.3.8.1.6 PCI Express Class Code Register—Offset 0x09

The class code register, shown in Figure 21-43, is comprised of three single-byte fields—base class (offset 0x0B), sub-class (offset 0x0A), and programming interface (offset 0x09)—that indicate the basic functionality of the function.

Offset 0x09 Access: Read only

	23–16	15–8	7–0
R	Base Class	Sub-Class	Programming Interface
W			
Reset	0 0 0 0 1 0 1 1	0 0 1 0 0 0 0 0	0x00 RC mode; 0x01 EP mode

Figure 21-43. PCI Express Class Code Register

Table 21-42 describes the class code register fields.

Table 21-42. PCI Express Class Code Register Field Descriptions

Bits	Name	Description
23–16	Base Class	0x0B—Processor
15–8	Sub-Class	0x20—PowerPC
7–0	Programming Interface	0x00—RC mode 0x01—EP mode

21.3.8.1.7 PCI Express Cache Line Size Register—Offset 0x0C

The cache line size register, shown in Figure 21-44, is provided for legacy compatibility purposes (PCI 2.3); it is not used for PCI Express device functionality.

Offset 0x0C Access: Read/Write

	7–0
R	Cache Line Size
W	
Reset	All zeros

Figure 21-44. PCI Express Bus Cache Line Size Register

Table 21-43 describes the cache line size register.

Table 21-43. PCI Express Bus Cache Line Size Register Field Descriptions

Bits	Name	Description
7–0	Cache Line Size	Represents the cache line size of the processor in terms of 32-bit words (8 32-bit words = 32 bytes). Note that for PCI Express operation this register is ignored.

21.3.8.1.8 PCI Express Latency Timer Register—0x0D

The latency timer register, shown in Figure 21-45, is provided for legacy compatibility purposes (PCI 2.3); it is not used for PCI Express device functionality.

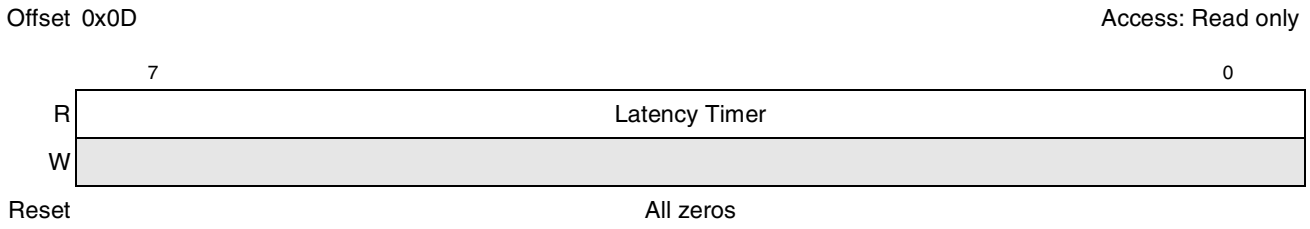


Figure 21-45. PCI Express Bus Latency Timer Register

Table 21-44 describes the PCI Express latency timer register (PLTR).

Table 21-44. PCI Express Bus Latency Timer Register Field Descriptions

Bits	Name	Description
7–0	Latency Timer	Note that for PCI Express operation this register is ignored.

21.3.8.1.9 PCI Express Header Type Register—0x0E

The PCI Express header type register, shown in Figure 21-44, is used to identify the layout of the PCI compatible header.

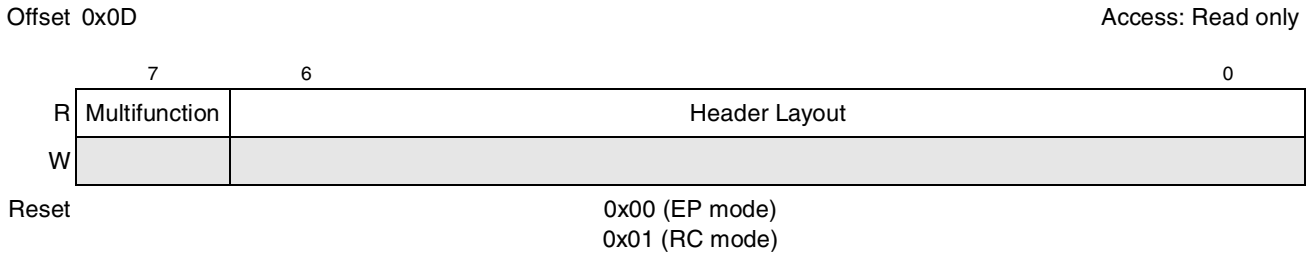


Figure 21-46. PCI Express Bus Latency Timer Register

Table 21-44 describes the PCI Express header type register.

Table 21-45. PCI Express Bus Latency Timer Register Field Descriptions

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 21-47 for type 0 layout. 0x01 Root Complex. See Figure 21-59 for type 1 layout. All other encodings reserved.

21.3.8.1.10 PCI Express BIST Register—0x0F

The BIST register is optional and reserved on the PCI Express controller.

21.3.8.2 Type 0 Configuration Header

The type 0 header is shown in Figure 21-47.

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
Base Address Registers				10 14 18 1C 20 24
				28
Subsystem ID		Subsystem Vendor ID		2C
				30
			Capabilities Pointer	34
Expansion ROM Base Address				38
MAX_LAT	MIN_GNT	Interrupt Pin	Interrupt Line	3C

Figure 21-47. PCI Express PCI-Compatible Configuration Header—Type 0

Section 21.3.8.1, "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 0x10.

21.3.8.2.1 PCI Express Base Address Registers—0x10–0x27

The PCI Express base address registers (BARs) point to the beginning of distinct address ranges which the device should claim. In EP mode, the device supports a configuration space BAR, a 32-bit memory space BAR, and two 64-bit memory space BARs. In RC mode, the device only supports the configuration space BAR in the header; the other memory spaces are defined by the inbound ATMUs. Refer to Section 21.3.5.2, "PCI Express Inbound ATMU Registers," for more information.

Base address register 0 at offset 0x10 is a special fixed 1M window that is used for inbound configuration accesses. This window is called the PCI Express configuration and status register base address register (PEXCSRBAR). Note that PEXCSRBAR cannot be updated through the inbound ATMU registers. The PEXCSRBAR is shown in Figure 21-48.

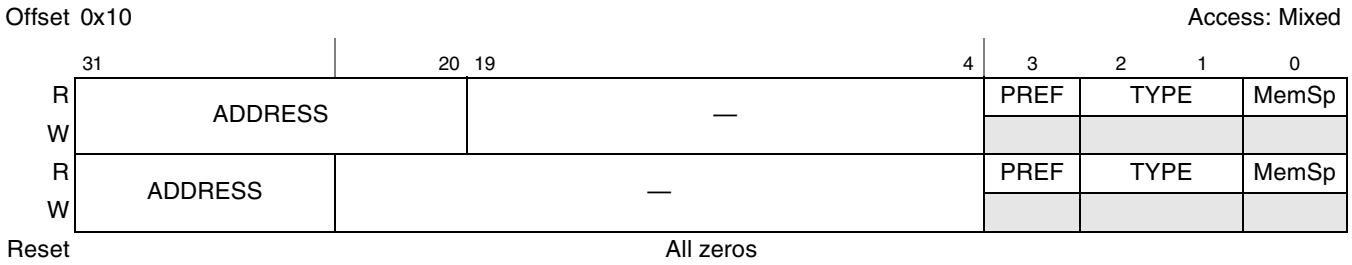


Figure 21-48. PCI Express Base Address Register 0 (PEXCSRBAR)

Table 21-46 describes the PCI Express configuration and status register base address register.

Table 21-46. PEXCSRBAR Field Descriptions

Bits	Name	Description
31–20	ADDRESS	Indicates the base address that the inbound configuration window occupies. This window is fixed at 1 Mbyte.
19–4	—	Reserved
31–24	ADDRESS	Indicates the base address that the inbound configuration window occupies. This window is fixed at 1 Mbyte.
23–4	—	Reserved
3	PREF	Prefetchable
2–1	TYPE	Type. 00 Locate anywhere in 32-bit address space.
0	MemSp	Memory space indicator

Base address register 1 at offset 0x14 is used to define the inbound memory window in the 32-bit memory space. The 32-bit memory BAR is shown in Figure 21-49.

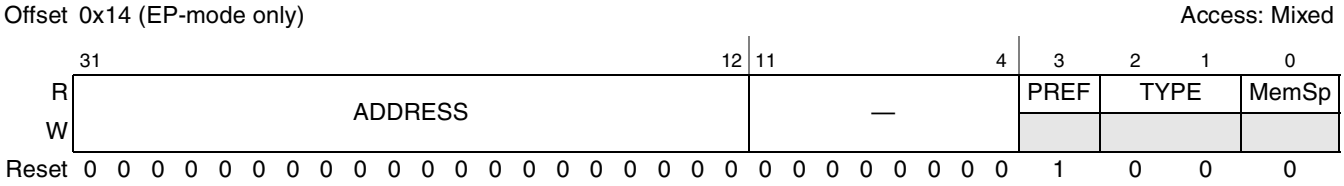


Figure 21-49. 32-Bit Memory Base Address Register (BAR1)

Table 21-47 describes the PCI Express 32-bit memory BAR fields.

Table 21-47. 32-Bit Memory Base Address Register (BAR1) Field Descriptions

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 inbound window size in the inbound window attributes register (PEXIWAR1).
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable. This bit is determined by PEXIWAR1[PF].

Table 21-47. 32-Bit Memory Base Address Register (BAR1) Field Descriptions (continued)

Bits	Name	Description
2–1	TYPE	Type. 00 Locate anywhere in 32-bit address space.
0	MemSp	Memory space indicator.

Base address register 2 at offset 0x18 and base address register 4 at offset 0x20 are used to define the lower portion of the 64-bit inbound memory windows. The 64-bit low memory BARs are shown in Figure 21-50.

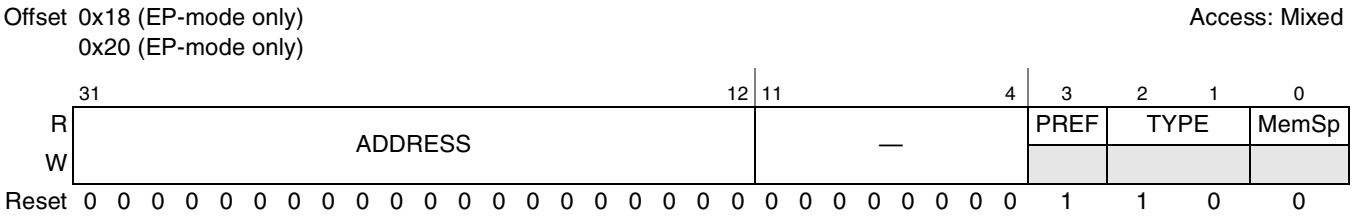


Figure 21-50. 64-Bit Low Memory Base Address Register

Table 21-48 describes the PCI Express 64-bit low memory BAR fields.

Table 21-48. 64-Bit Low Memory Base Address Register Field Descriptions

Bits	Name	Description
31–12	ADDRESS	Indicates the lower portion of the base address where the inbound memory window begins. The number of bits that the device allows to be writable is selected through the inbound window size in the inbound window attributes registers (PEXIWAR2 for offset 0x18 and PEXIWAR3 for offset 0x20).
11–4	—	Reserved. The device allows a 4 Kbyte window minimum.
3	PREF	Prefetchable. This bit is determined by PEXIWAR*n*[2].
2–1	TYPE	Type. 0b10 Locate anywhere in 64-bit address space.
0	MemSp	Memory space indicator

Base address register 3 at offset 0x1C and base address register 5 at offset 0x24 are used to define the upper portion of the 64-bit inbound memory windows. The 64-bit high memory BARs are shown in Figure 21-51.

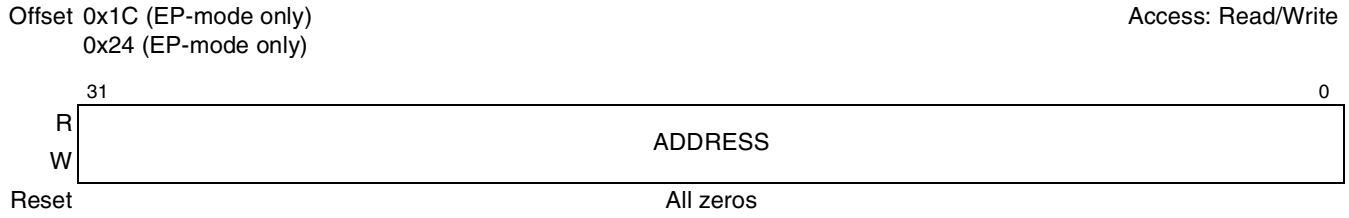


Figure 21-51. 64-Bit High Memory Base Address Register

Table 21-49 describes the PCI Express 64-bit low memory BAR fields.

Table 21-49. Bit Setting for 64-Bit High Memory Base Address Register

Bits	Name	Description
31–0	ADDRESS	Indicates the upper portion of the base address where the inbound memory window begins. The number of bits that the device allows to be writable is selected through the inbound window size in the inbound window attributes registers (PEXIWAR2 for offset 0x1C and PEXIWAR3 for offset 0x24). If no access to local memory is to be permitted by external requestors, then all bits are programmed.

21.3.8.2.2 PCI Express Subsystem Vendor ID Register (EP-Mode Only)—0x2C

The PCI Express subsystem vendor ID register is used to identify the subsystem.

Offset 0x2C (EP-mode only) Access: Read only

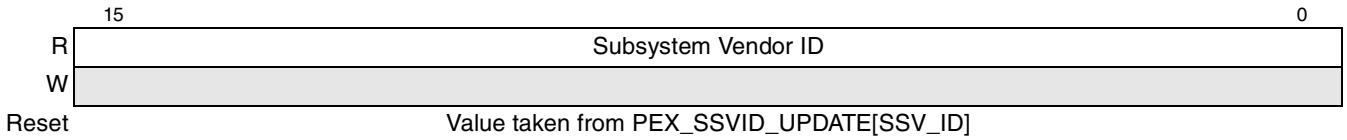


Figure 21-52. PCI Express Subsystem Vendor ID Register

Table 21-50. PCI Express Subsystem Vendor ID Register Field Description

Bits	Name	Description
15–0	Subsystem Vendor ID	The value for subsystem vendor ID is determined by the PCI Express subsystem vendor ID update register. See Section 21.3.10.17, “PCI Express Subsystem Vendor ID Update Register (EP Mode Only)—0x478,” for more information.

21.3.8.2.3 PCI Express Subsystem ID Register (EP-Mode Only)—0x2E

The PCI Express subsystem ID register is used to identify the subsystem.

Offset 0x2E (EP-mode only) Access: Read only

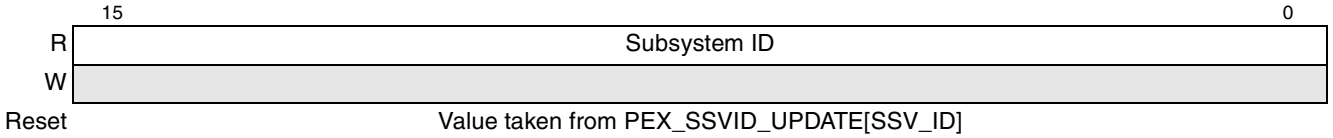


Figure 21-53. PCI Express Subsystem ID Register

Table 21-51. PCI Express Subsystem ID Register Field Description

Bits	Name	Description
15–0	Subsystem ID	The value for subsystem ID is determined by the PCI Express subsystem vendor ID update register. See Section 21.3.10.17, “PCI Express Subsystem Vendor ID Update Register (EP Mode Only)—0x478,” for more information.

21.3.8.2.4 Capabilities Pointer Register—0x34

The capabilities pointer identifies additional functionality supported by the device.

Offset 0x34 Access: Read only

	7						0
R	Capabilities Pointer						
W							
Reset	0	1	0	0	0	1	0	0

Figure 21-54. Capabilities Pointer Register

Table 21-52. Capabilities Pointer Register Field 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 21.3.9, "PCI Compatible Device-Specific Configuration Space," for more information.

21.3.8.2.5 PCI Express Interrupt Line Register (EP-Mode Only)—0x3C

The 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 0x3C (EP-mode only) Access: Read/Write

	7	0
R	Interrupt Line	
W		
Reset	All zeros	

Figure 21-55. PCI Express Interrupt Line Register

Table 21-53. PCI Express Interrupt Line Register Field Description

Bits	Name	Description
7–0	Interrupt Line	Used to communicate interrupt line routing information.

21.3.8.2.6 PCI Express Interrupt Pin Register—0x3D

The interrupt pin register identifies the legacy interrupt (INTx) messages the device (or function) uses.

Offset 0x3D Access: Read only

	7						0
R	Interrupt Pin						
W							
Reset	0	0	0	0	0	0	0	1

Figure 21-56. PCI Express Interrupt Pin Register

Table 21-54. PCI Express Interrupt Pin Register Field Description

Bits	Name	Description
7–0	Interrupt pin	Legacy INTx message used by this device. 0x00 This device does not use legacy interrupt (INTx) messages. 0x01 INTA 0x02 INTB 0x03 INTC 0x04 INTD all others Reserved.

21.3.8.2.7 PCI Express Minimum Grant Register (EP-Mode Only)—0x3E

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x3E (EP-mode only) Access: Read only

	7	0
R	MIN_GNT	
W		
Reset	All zeros	

Figure 21-57. PCI Express Maximum Grant Register (MAX_GNT)

Table 21-55. PCI Express Maximum Grant Register Field Description

Bits	Name	Description
7–0	MIN_GNT	Does not apply for PCI Express.

21.3.8.2.8 PCI Express Maximum Latency Register (EP-Mode Only)—0x3F

This register does not apply to PCI Express. It is present for legacy purposes.

Offset 0x3F (EP-mode only) Access: Read only

	7	0
R	MAX_LAT	
W		
Reset	All zeros	

Figure 21-58. PCI Express Maximum Latency Register (MAX_LAT)

Table 21-56. PCI Express Maximum Latency Register Field Description

Bits	Name	Description
7–0	MAX_LAT	Does not apply for PCI Express.

21.3.8.3 Type 1 Configuration Header

The type 1 header is shown in Figure 21-59.

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
Base Address Register 0				10
				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
			Capabilities Pointer	34
Bridge Control		Interrupt Pin	Interrupt Line	3C

Figure 21-59. PCI Express PCI-Compatible Configuration Header—Type 1

Section 21.3.8.1, “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 0x10.

21.3.8.3.1 PCI Express Base Address Register 0—0x10

Base address register 0 at offset 0x10 is a special fixed 1-Mbyte window that is used for inbound configuration accesses. This window is called the PCI Express configuration and status register base address register (PEXCSRBAR). Note that PEXCSRBAR cannot be updated through the inbound ATMU registers. The PEXCSRBAR is shown in Figure 21-48.

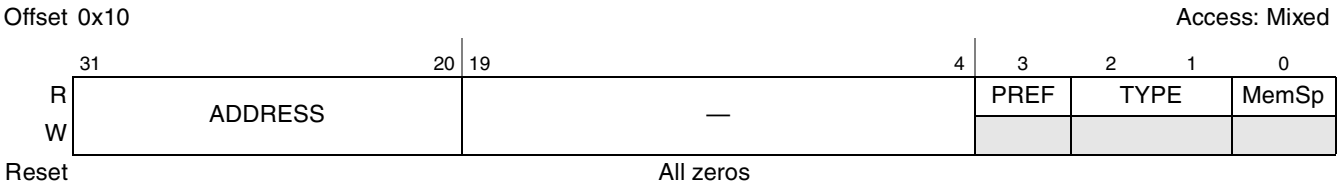


Figure 21-60. PCI Express Base Address Register 0 (PEXCSRBAR)

Table 21-46 describes the PCI Express configuration and status register base address register.

Table 21-57. PEXCSRBAR Field Descriptions

Bits	Name	Description
31–20	ADDRESS	Indicates the base address that the inbound configuration window occupies. This window is fixed at 1 Mbyte.
19–4	—	Reserved
3	PREF	Prefetchable
2–1	TYPE	Type. 00 Locate anywhere in 32-bit address space.
0	MemSp	Memory space indicator

21.3.8.3.2 PCI Express Primary Bus Number Register—Offset 0x18

The primary bus number register is shown in Figure 21-61.

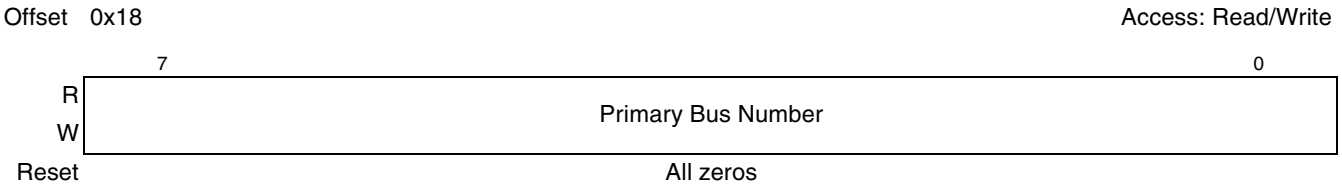


Figure 21-61. PCI Express Primary Bus Number Register

Table 21-58 describes the primary bus number register fields.

Table 21-58. PCI Express Primary Bus Number Register Field 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.

21.3.8.3.3 PCI Express Secondary Bus Number Register—Offset 0x19

The secondary bus number register is shown in Figure 21-62.

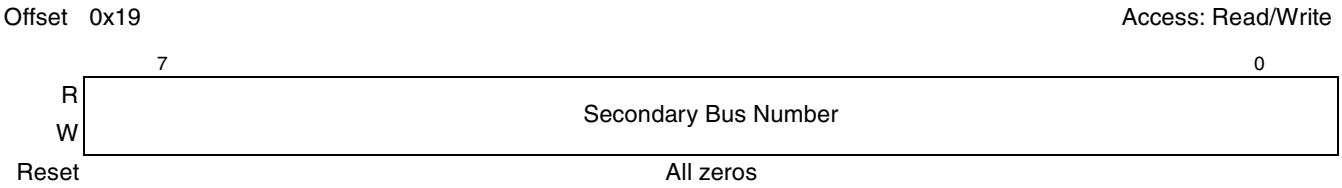


Figure 21-62. PCI Express Secondary Bus Number Register

Table 21-59 describes the secondary bus number register fields.

Table 21-59. PCI Express Secondary Bus Number Register Field 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.

21.3.8.3.4 PCI Express Subordinate Bus Number Register—Offset 0x1A

The subordinate bus number register is shown in Figure 21-63.

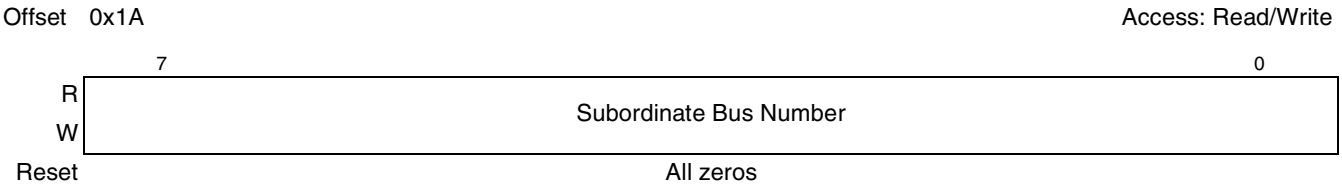


Figure 21-63. PCI Express Subordinate Bus Number Register

Table 21-60 describes the subordinate bus number register fields.

Table 21-60. PCI Express Subordinate Bus Number Register Field Description

Bits	Name	Description
7–0	Subordinate Bus Number	Highest bus number that is on the downstream interface.

21.3.8.3.5 PCI Express Secondary Latency Timer Register—0x1B

The secondary latency timer register does not apply to PCI Express. It must be read-only and return all zeros when read.

21.3.8.3.6 PCI Express I/O Base Register—0x1C

Note that this device does not support inbound I/O transactions. The I/O base register is shown in Figure 21-63.

Offset 0x1C Access: Read only

	7	4	3	0
R	I/O Start Address		Address Decode Type	
W				
Reset	All zeros			

Figure 21-64. PCI Express I/O Base Register

Table 21-60 describes the I/O base register fields.

Table 21-61. PCI Express I/O Base Register Field 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.

21.3.8.3.7 PCI Express I/O Limit Register—0x1D

Note that this device does not support inbound I/O transactions. The I/O limit register is shown in Figure 21-63.

Offset 0x1D Access: Read only

	7	4	3	0
R	I/O Limit Address		Address Decode Type	
W				
Reset	All zeros			

Figure 21-65. PCI Express I/O Limit Register

Table 21-60 describes the I/O limit register fields.

Table 21-62. PCI Express I/O Limit Register Field 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.

21.3.8.3.8 PCI Express Secondary Status Register—0x1E

The PCI Express secondary status register is shown in Figure 21-66. Note that the errors in this register may be masked by corresponding bits in the secondary status interrupt mask register (PEX_SS_INTR_MASK) and that by default all of the errors are masked. See Section 21.3.10.20, "Secondary Status Interrupt Mask Register (RC-Mode Only)—0x5A0," for more information.

Offset 0x1E 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 21-66. PCI Express Secondary Status Register

Table 21-63 describes the PCI Express secondary status register fields.

Table 21-63. PCI Express Secondary Status Register Field Description

Bits	Name	Description
15	DPE	Detected parity error. This bit is set whenever 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, provided 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

21.3.8.3.9 PCI Express Memory Base Register—0x20

The memory base register is shown in Figure 21-67.

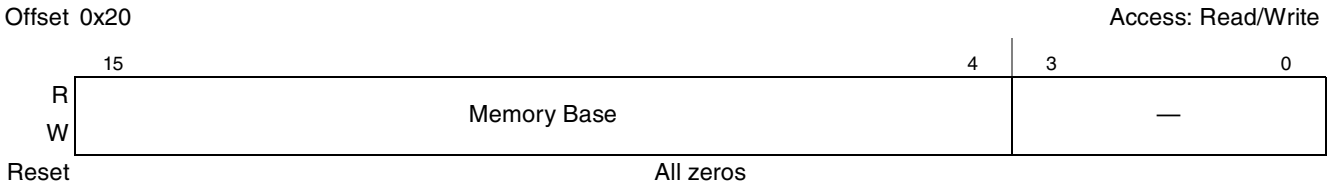


Figure 21-67. PCI Express Memory Base Register

Table 21-64 describes the memory base register fields.

Table 21-64. PCI Express Memory Base Register Field 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

21.3.8.3.10 PCI Express Memory Limit Register—0x22

The memory limit register is shown in Figure 21-68.

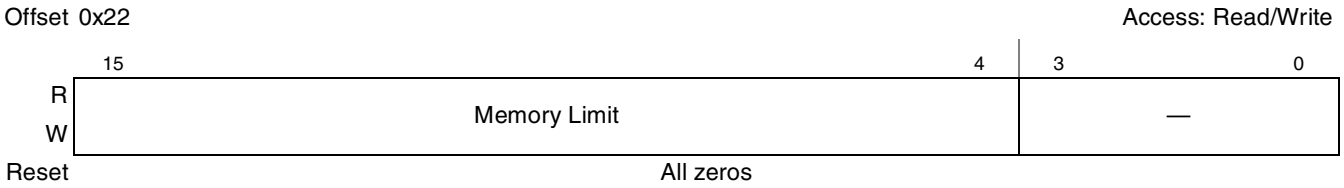


Figure 21-68. PCI Express Memory Limit Register

Table 21-65 describes the memory base register fields.

Table 21-65. PCI Express Memory Limit Register Field 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 results in unsupported request response.
3–0	—	Reserved

21.3.8.3.11 PCI Express Prefetchable Memory Base Register—0x24

The prefetchable memory base register is shown in Figure 21-69.

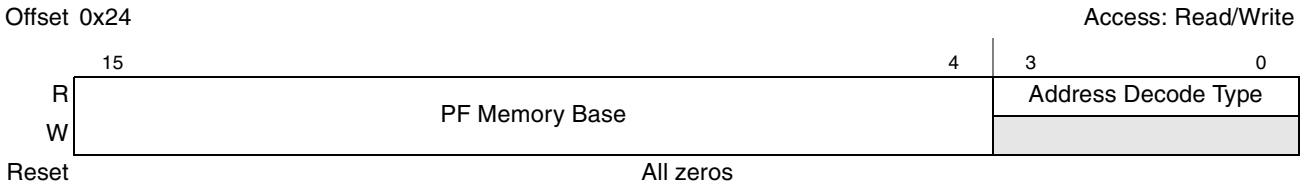


Figure 21-69. PCI Express Prefetchable Memory Base Register

Table 21-66 describes the prefetchable memory base register fields.

Table 21-66. PCI Express Prefetchable Memory Base Register Field 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	Specifies the number of prefetchable memory address bits. 0x00 32-bit memory address decode 0x01 64-bit memory address decode All other settings reserved.

21.3.8.3.12 PCI Express Prefetchable Memory Limit Register—0x26

The prefetchable memory limit register is shown in Figure 21-70.

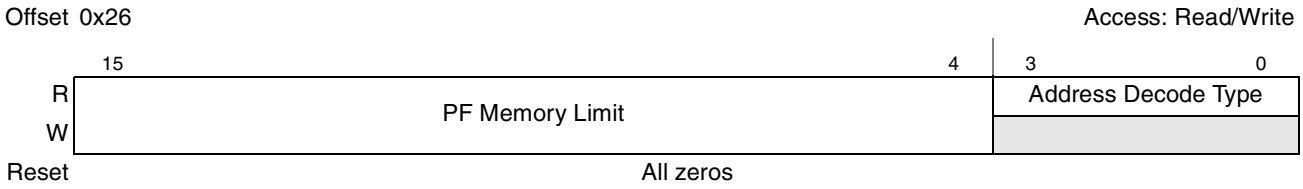


Figure 21-70. PCI Express Prefetchable Memory Limit Register

Table 21-67 describes the prefetchable memory limit register fields.

Table 21-67. PCI Express Prefetchable Memory Limit Register Field 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.

21.3.8.3.13 PCI Express Prefetchable Base Upper 32 Bits Register—0x28

The PCI Express prefetchable memory base upper 32 bits register is shown in Figure 21-71.

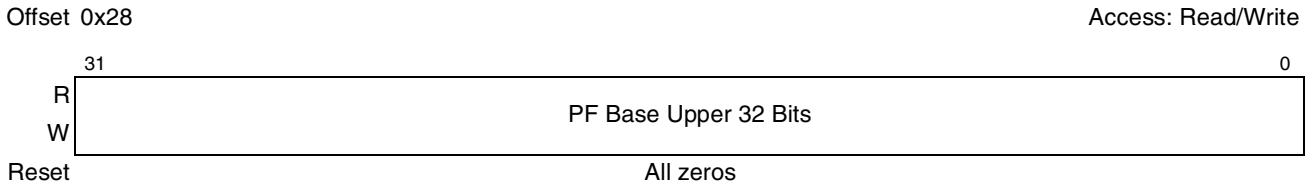


Figure 21-71. PCI Express Prefetchable Base Upper 32 Bits Register

Table 21-68 describes the PCI Express prefetchable memory base upper 32 bits register fields.

Table 21-68. PCI Express Prefetchable Base Upper 32 Bits Register

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.

21.3.8.3.14 PCI Express Prefetchable Limit Upper 32 Bits Register—0x2C

The PCI Express prefetchable memory base upper 32 bits register is shown in Figure 21-72.

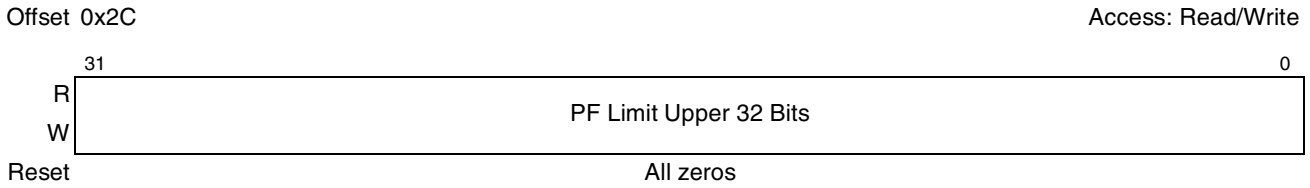


Figure 21-72. PCI Express Prefetchable Limit Upper 32 Bits Register

Table 21-69 describes the PCI Express prefetchable memory limit upper 32 bits register fields.

Table 21-69. PCI Express Prefetchable Limit Upper 32 Bits Register

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.

21.3.8.3.15 PCI Express I/O Base Upper 16 Bits Register—0x30

Note that this device does not support inbound I/O transactions. The I/O base upper 16 bits register is shown in Figure 21-73.

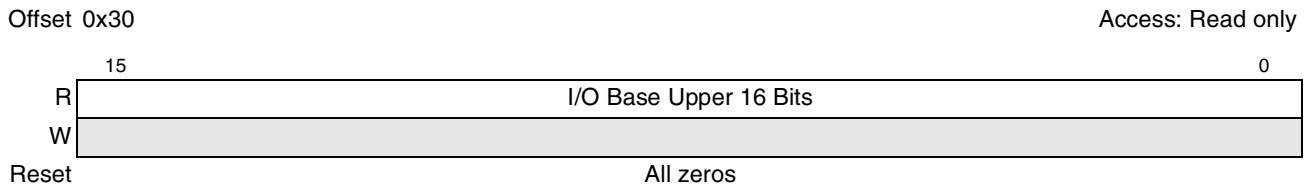


Figure 21-73. PCI Express I/O Base Upper 16 Bits Register

Table 21-70 describes the I/O base upper 16 bits register fields.

Table 21-70. PCI Express I/O Base Upper 16 Bits Register Field 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.

21.3.8.3.16 PCI Express I/O Limit Upper 16 Bits Register—0x32

Note that this device does not support inbound I/O transactions. The I/O limit upper 16 bits register is shown in Figure 21-74.

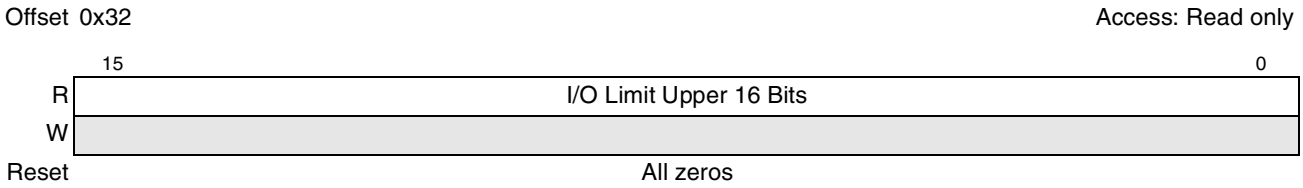


Figure 21-74. PCI Express I/O Limit Upper 16 Bits Register

Table 21-71 describes the I/O limit upper 16 bits register fields.

Table 21-71. PCI Express I/O Limit Upper 16 Bits Register Field 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.

21.3.8.3.17 Capabilities Pointer Register—0x34

The capabilities pointer identifies additional functionality supported by the device.

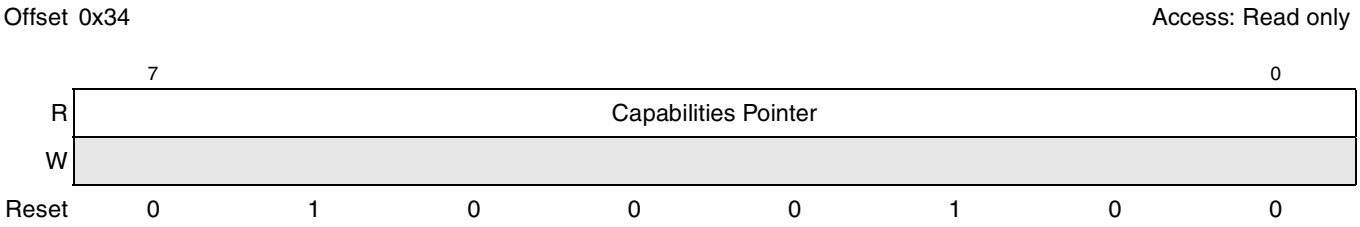


Figure 21-75. Capabilities Pointer Register

Table 21-72. Capabilities Pointer Register Field 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 21.3.9, “PCI Compatible Device-Specific Configuration Space,” for more information.

21.3.8.3.18 PCI Express Interrupt Line Register—0x3C

The 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.

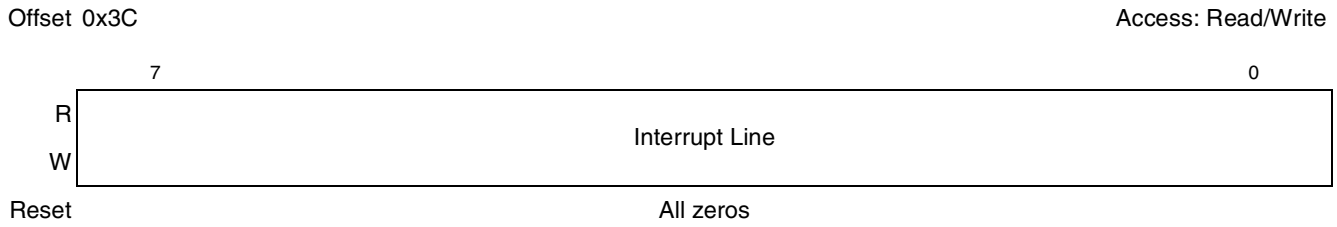


Figure 21-76. PCI Express Interrupt Line Register

Table 21-73. PCI Express Interrupt Line Register Field Description

Bits	Name	Description
7–0	Interrupt Line	Used to communicate interrupt line routing information.

21.3.8.3.19 PCI Express Interrupt Pin Register—0x3D

The interrupt pin register identifies the legacy interrupt (INTx) messages the device (or function) uses.

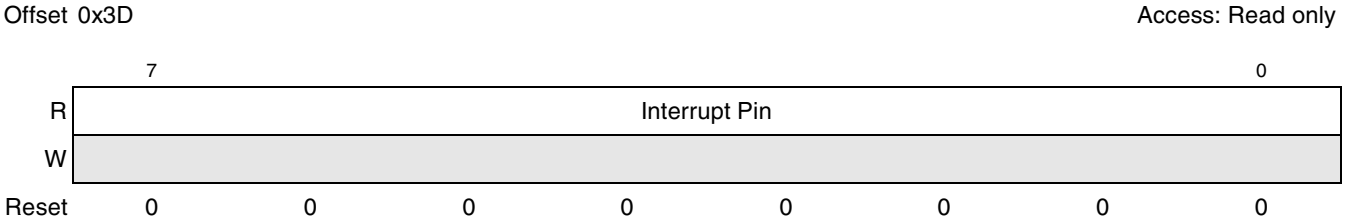


Figure 21-77. PCI Express Interrupt Pin Register

Table 21-74. PCI Express Interrupt Pin Register Field Description

Bits	Name	Description
7–0	Interrupt pin	Legacy INTx message used by this device. 0x00 This device does not use legacy interrupt (INTx) messages. 0x01 INTA 0x02 INTB 0x03 INTC 0x04 INTD all others Reserved.

21.3.8.3.20 PCI Express Bridge Control Register—0x3E

The PCI Express bridge control register is shown in Figure 21-78.

Offset 0x3E Access: Read/Write

	15–7	6	5–4	3	2	1	0
R W	—	Scnd_RST	—	VGA_EN	ISA_EN	SERR_EN	PER
Reset	All zeros						

Figure 21-78. PCI Express Bridge Control Register

Table 21-75 describes the PCI Express bridge control register fields.

Table 21-75. PCI Express Bridge Control Register Field 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. This bit controls the propagation of ERR_COR, ERR_NONFATAL, and ERR_FATAL responses received on the secondary side.
0	PER	Parity error response.

21.3.9 PCI Compatible Device-Specific Configuration Space

The PCI compatible device-specific configuration space is a PCI compatible configuration space from 0x40 to 0xFF (just above the 64-byte PCI-compatible configuration header).

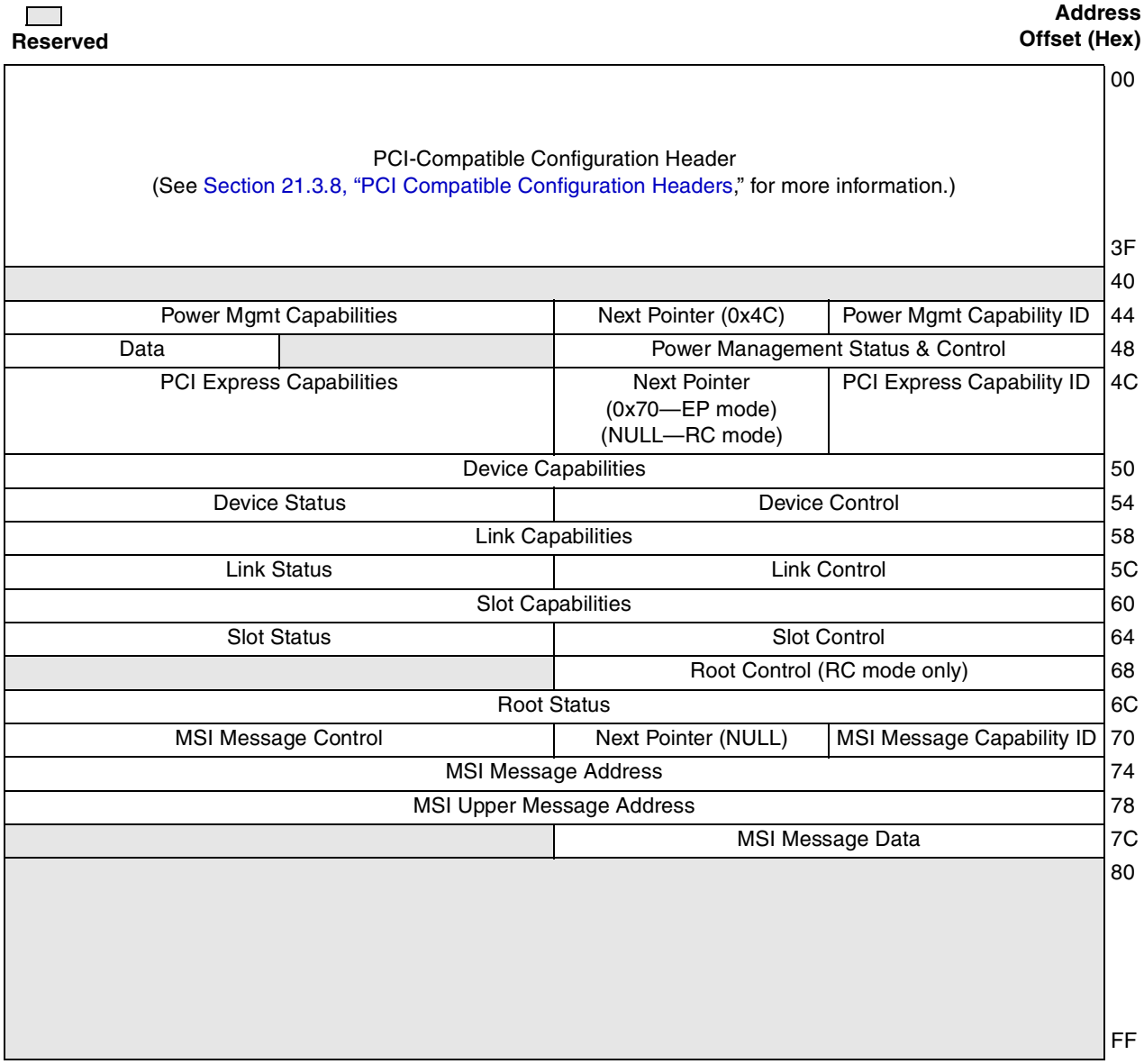


Figure 21-79. PCI Compatible Device-Specific Configuration Space

21.3.9.1 PCI Express Power Management Capability ID Register—0x44

The PCI Express power management capability ID register is shown in Figure 21-80.

Offset 0x44 Access: Read only

	7							0
R	Power Mgmt Capability ID							
W								
Reset	0	0	0	0	0	0	0	1

Figure 21-80. PCI Express Power Management Capability ID Register

Table 21-76. PCI Express Power Management Capability ID Register Field Description

Bits	Name	Description
7–0	Power Mgmt Capability ID	Power Management = 0x01

21.3.9.2 PCI Express Power Management Capabilities Register—0x46

The PCI Express power management capabilities register is shown in Figure 21-81.

Offset 0x46 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 21-81. PCI Express Power Management Capabilities Register

Table 21-77. PCI Express Power Management Capabilities Register Field Description

Bits	Name	Description
15–11	PME Support	Indicates the power states that this device supports
10	D2	D2 Support
9	D1	D1 Support
8–6	AUX Curr	AUX Current
5	DSI	Device Specific Initialization
4	—	Reserved
3	PME CLK	Does not apply to PCI Express.
2–0	Version	Set to 0x2 for this version of the specification.

21.3.9.3 PCI Express Power Management Status and Control Register—0x48

The PCI Express power management status and control register is shown in Figure 21-82.

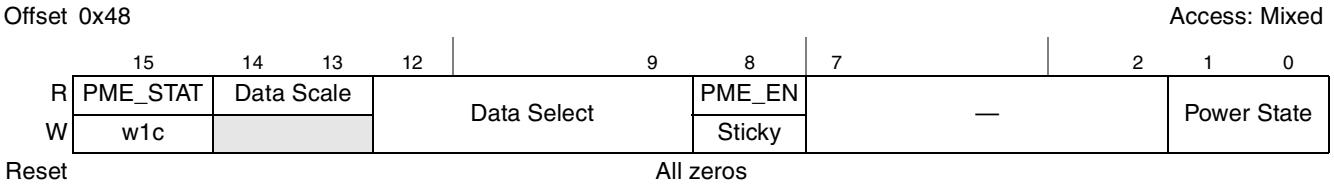


Figure 21-82. PCI Express Power Management Status and Control Register

Table 21-78. PCI Express Status and Control Register Field Description

Bits	Name	Description
15	PME_STAT	PME Status
14–13	Data Scale	Obtained directly from *PCI Express*™ *Base Specification, Revision 1.0a* .
12–9	Data Select	Obtained directly from *PCI Express*™ *Base Specification, Revision 1.0a* .
8	PME_EN	PME Enable
7–2	—	Reserved
1–0	Power State	Power state. Indicates the current power state of the function. 0x00 D0 0x01 D1 0x02 D2 0x03 D3

21.3.9.4 PCI Express Power Management Data Register—0x4B

The PCI Express power management data register is shown in Figure 21-83.

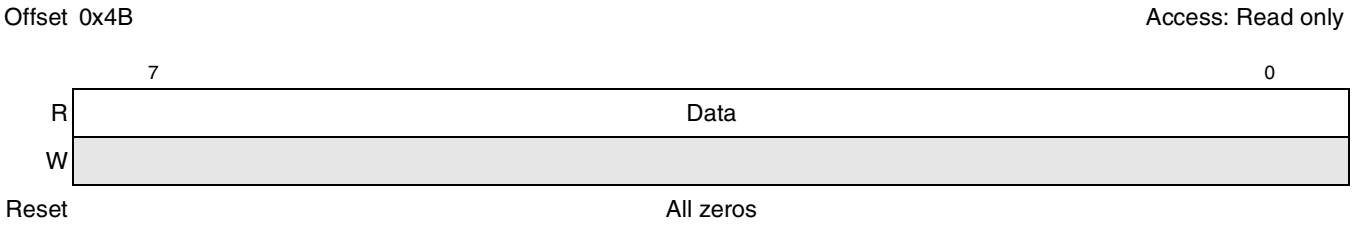


Figure 21-83. PCI Express Power Management Data Register

Table 21-79. PCI Express Power Management Data Register Field Description

Bits	Name	Description
7–0	Data	Obtained from *PCI Express*™ *Base Specification, Revision 1.0a* .

21.3.9.5 PCI Express Capability ID Register—0x4C

The PCI Express capability ID register is shown in Figure 21-84.

Offset 0x4C Access: Read only

	7							0
R	PCI Express Capability ID							
W								
Reset	0	0	0	1	0	0	0	0

Figure 21-84. PCI Express Capability ID Register

Table 21-80. PCI Express Capability ID Register Field Description

Bits	Name	Description
7–0	PCI Express Capability ID	PCI Express = 0x10

21.3.9.6 PCI Express Capabilities Register—0x4E

The PCI Express capabilities register is shown in Figure 21-85.

Offset 0x4E Access: Read only

	15	14	13					8	7			4	3			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 21-85. PCI Express Capabilities Register

Table 21-81. PCI Express Capabilities Register Field Description

Bits	Name	Description
15–14	—	Reserved
13–9	Interrupt Message Number	If this function is allocated more than one MSI interrupt number, then this register is required to contain the offset between the base Message Data and the MSI Message that is generated when any of the status bits in either the Slot Status register or the Root Port Status register, of this capability structure, are set.
8	Slot	Slot Implemented (RC mode only)
7–4	Device/Port Type	0100 (RC mode) 0000 (EP mode)
3–0	Capability Version	Indicates the defined PCI Express capability structure version number. Must be 1h for 1.0, 1.0a, and 1.1 specification.

21.3.9.7 PCI Express Device Capabilities Register—0x50

The PCI Express device capabilities register is shown in Figure 21-86.

Offset 0x50 Access: Read only

	31–28	27–26	25–18	17–16
R	—	CSPLS	CSPLV	—
W				
Reset	0 0 0 0	0 0	1 1 1 1 1 1 1 1	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 1

Figure 21-86. PCI Express Device Capabilities Register

Table 21-82. PCI Express Device Capabilities Register Field 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. 001 = 256-bytes

21.3.9.8 PCI Express Device Control Register—0x54

The PCI Express device control register is shown in Figure 21-87.

Offset 0x54 Access: Read/write

	15	14–12	11	10	9	8	7–5	4	3	2	1	0
R	—	MAX_READ_SIZE	NSE	APE	PFE	ETE	MAX_PAYLOAD_SIZE	RO	URR	FER	NFER	CER
W												
Reset	0	0 1 0	1	0	0	0	0 0 0	1	0	0	0	0

Figure 21-87. PCI Express Device Control Register

Table 21-83. PCI Express Device Control Register Field 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

21.3.9.9 PCI Express Device Status Register—0x56

The PCI Express device status register is shown in Figure 21-88.

Offset 0x56 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 21-88. PCI Express Device Status Register

Table 21-84. PCI Express Device Status Register Field 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

21.3.9.10 PCI Express Link Capabilities Register—0x58

The PCI Express link capabilities register is shown in Figure 21-89.

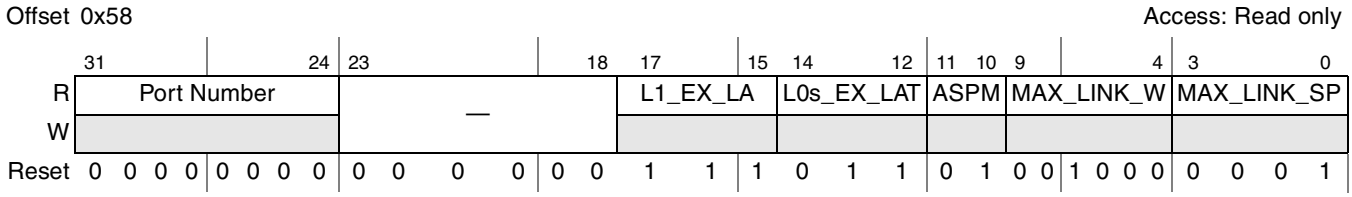


Figure 21-89. PCI Express Link Capabilities Register

Table 21-85. PCI Express Link Capabilities Register Field Description

Bits	Name	Description
31–24	Port Number	—
23–18	—	Reserved
17–15	L1_EX_LAT	L1 exit latency
14–12	L0s_EX_LAT	L0s exit latency
11–10	ASPM	Active state power management (ASPM) Support
9–4	MAX_LINK_W	Maximum link width
3–0	MAX_LINK_SP	Maximum link speed 0001 2.5 GT/s link

21.3.9.11 PCI Express Link Control Register—0x5C

The PCI Express link control register is shown in Figure 21-90.

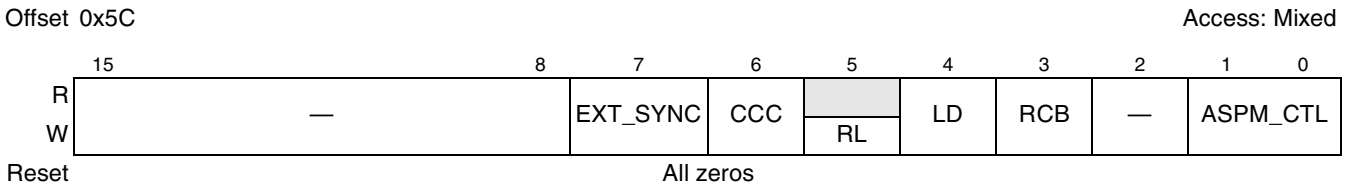


Figure 21-90. PCI Express Link Control Register

Table 21-86. PCI Express Link Control Register Field Description

Bits	Name	Description
15–8	—	Reserved
7	EXT_SYNC	Extended synch
6	CCC	Common clock configuration
5	RL	Retrain link (Reserved for EP devices). In RC mode, setting this bit initiates link retraining by directing the Physical Layer LTSSM to the Recovery state; reads of this bit always return 0.

Table 21-86. PCI Express Link Control Register Field Description (continued)

Bits	Name	Description
4	LD	Link disable (Reserved for EP devices)
3	RCB	Read completion boundary
2	—	Reserved
1–0	ASPM_CTL	Active state power management (ASPM) control

21.3.9.12 PCI Express Link Status Register—0x5E

The PCI Express link status register is shown in Figure 21-91.

Offset 0x5E 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 21-91. PCI Express Link Status Register

Table 21-87. PCI Express Link Status Register Field 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.

21.3.9.13 PCI Express Slot Capabilities Register—0x60

The PCI Express slot capabilities register is shown in Figure 21-92.

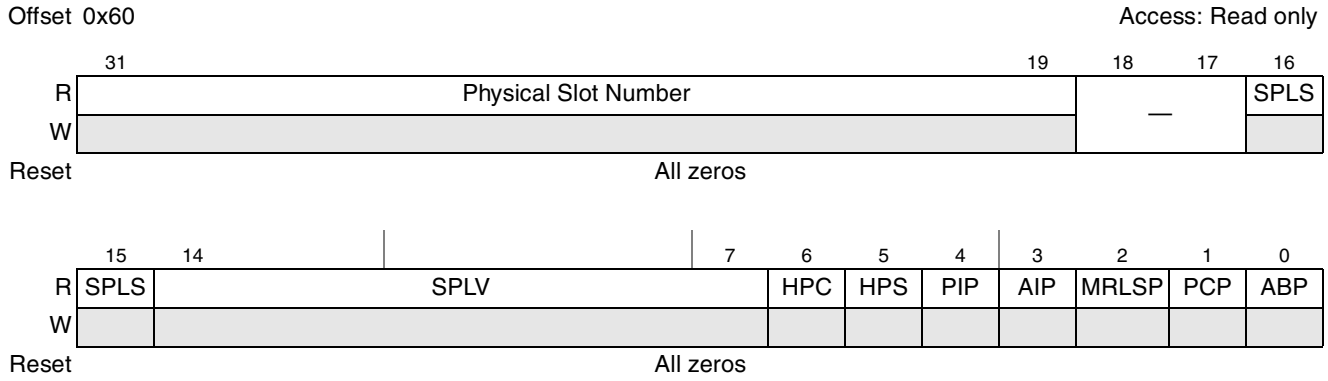


Figure 21-92. PCI Express Slot Capabilities Register

Table 21-88. PCI Express Slot Capabilities Register Field Description

Bits	Name	Description
31–19	Physical Slot Number	This hardware initialized field indicates the physical slot number attached to this Port. This field must be hardware initialized to a value that assigns a slot number that is globally unique within the chassis. These registers should be initialized to 0 for Ports connected to devices that are either integrated on the system board or integrated within the same silicon as the Switch device or Root Port.
18–17	—	Reserved
16–15	SPLS	Slot power limit scale.
14–7	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.

21.3.9.14 PCI Express Slot Control Register—0x64

The PCI Express slot control register is shown in Figure 21-93.

Offset 0x64 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 21-93. PCI Express Slot Control Register

Table 21-89. PCI Express Slot Control Register Field 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.

21.3.9.15 PCI Express Slot Status Register—0x66

The PCI Express slot status register is shown in Figure 21-94.

Offset 0x66 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 21-94. PCI Express Slot Status Register

Table 21-90. PCI Express Slot Status Register Field Descriptions

Bits	Name	Description
15–7	—	Reserved
6	PDS	Presence detect state. This bit indicates the presence of a card in the slot. 0 Slot empty 1 Card is present

Table 21-90. PCI Express Slot Status Register Field Descriptions (continued)

Bits	Name	Description
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.

21.3.9.16 PCI Express Root Control Register (RC Mode Only)—0x68

The PCI Express root control register is shown in Figure 21-95.

Offset 0x68 Access: Read/Write

	15–4	3	2	1	0
R W	—	PMEIE	SEFEE	SENFEE	SECEE
Reset	All zeros				

Figure 21-95. PCI Express Root Control Register

Table 21-91. PCI Express Root Control Register Field 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.

21.3.9.17 PCI Express Root Status Register (RC Mode Only)—0x6C

The PCI Express root status register is shown in Figure 21-96.

Offset 0x6C 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 21-96. PCI Express Root Status Register

Table 21-92. PCI Express Root Status Register Field Description

Bits	Name	Description
31–18	—	Reserved
17	PMEP	PME pending.
16	PMES	PME status.
15–0	PME Requestor ID	PME requestor ID.

21.3.9.18 PCI Express MSI Message Capability ID Register (EP Mode Only)—0x70

The PCI Express MSI message capability ID register is shown in Figure 21-97.

Offset 0x70 Access: Read only

	7–0
R	MSI Message Capability ID
W	
Reset	0 0 0 0 0 1 0 1

Figure 21-97. PCI Express Capability ID Register

Table 21-93. PCI Express Capability ID Register Field Description

Bits	Name	Description
7–0	MSI Message Capability ID	MSI Message = 0x05

21.3.9.19 PCI Express MSI Message Control Register (EP Mode Only)—0x72

The PCI Express MSI message control register is shown in Figure 21-98.

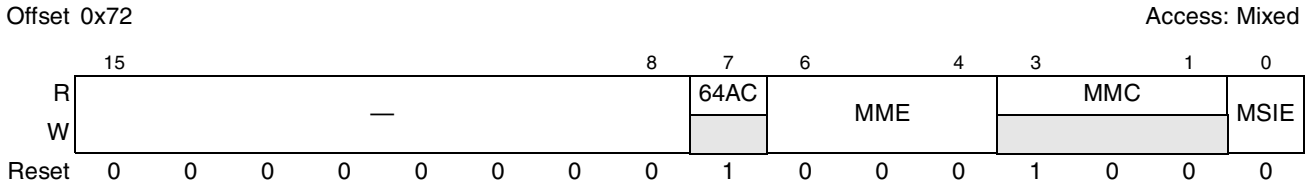


Figure 21-98. PCI Express MSI Message Control Register

Table 21-94. PCI Express MSI Message Control Register Field 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.

21.3.9.20 PCI Express MSI Message Address Register (EP Mode Only)—0x74

The PCI Express MSI message address register is shown in Figure 21-99.

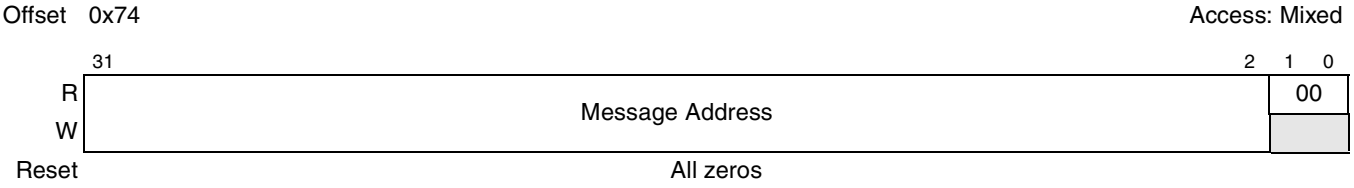


Figure 21-99. PCI Express MSI Message Address Register

Table 21-95. PCI Express MSI Message Address Register Field 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.

21.3.9.21 PCI Express MSI Message Upper Address Register (EP Mode Only)—0x78

The PCI Express MSI message upper address register is shown in Figure 21-100.

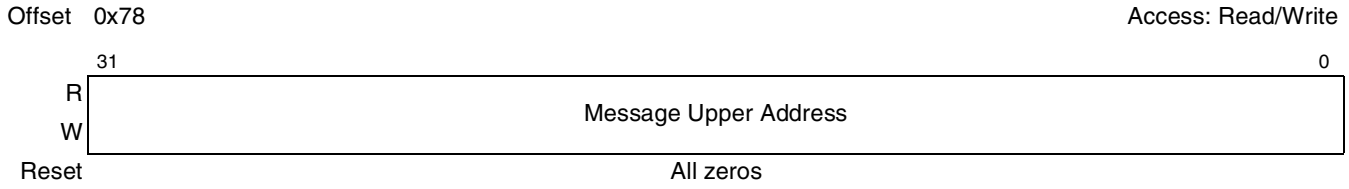


Figure 21-100. PCI Express MSI Message Upper Address Register

Table 21-96. PCI Express MSI Message Upper Address Register Field Description

Bits	Name	Description
31–0	Message Upper Address	System-specified message upper address

21.3.9.22 PCI Express MSI Message Data Register (EP Mode Only)—0x7C

The PCI Express MSI message data register is shown in Figure 21-101.

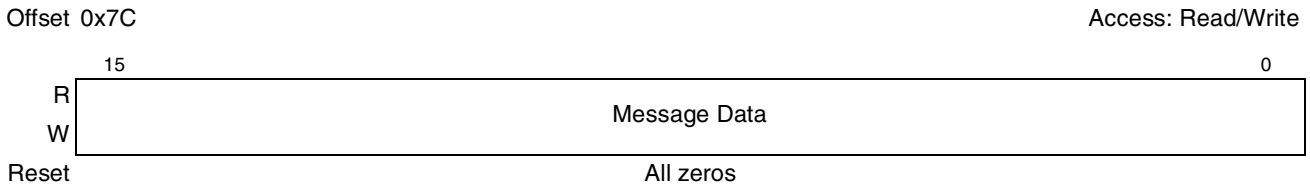


Figure 21-101. PCI Express MSI Message Data Register

Table 21-97. PCI Express MSI Message Data Register Field Description

Bits	Name	Description
15–0	Message Data	System-specified message.

21.3.10 PCI Express Extended Configuration Space

Reserved

		Address Offset (Hex)
PCI Compatible Configuration Header (See Section 21.3.8, “PCI Compatible Configuration Headers,” for more information.)		000 03F
PCI-Compatible Device-Specific Configuration Space (See Section 21.3.9, “PCI Compatible Device-Specific Configuration Space,” for more information.)		040 0FF
Next Capability Offset (NULL)/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[1]		400 6FF
		700 FFF

Figure 21-102. PCI Express Extended Configuration Space

1 Note that the PCI Express Controller Internal CSRs are not accessible by inbound PCI Express configuration transactions. Attempts to access these registers returns all 0s.

21.3.10.1 PCI Express Advanced Error Reporting Capability ID Register—0x100

The PCI Express advanced error reporting capability ID register is shown in Figure 21-103.

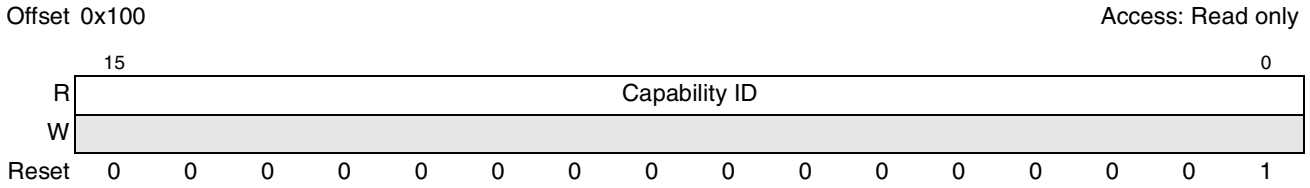


Figure 21-103. PCI Express Advanced Error Reporting Capability ID Register

Table 21-98. PCI Express Advanced Error Reporting Capability ID Register Field Description

Bits	Name	Description
15–0	Capability ID	Advanced error reporting capability = 0x0001

21.3.10.2 PCI Express Uncorrectable Error Status Register—0x104

The PCI Express uncorrectable error status register is shown in Figure 21-104.

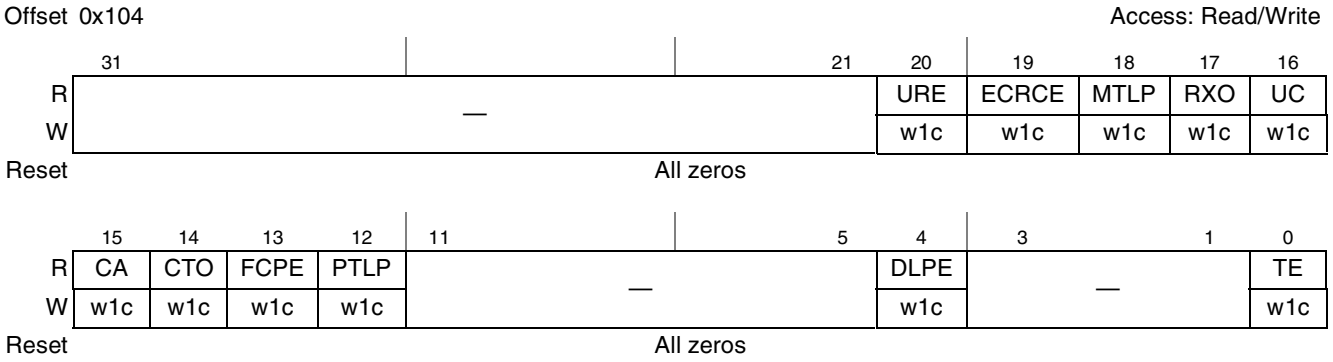


Figure 21-104. PCI Express Uncorrectable Error Status Register

Table 21-99. PCI Express Uncorrectable Error Status Register Field 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. Note that a completion timeout error is a fatal error. If a completion timeout error is detected, the system has become unstable. Hot reset is recommended to restore stability of the system.

Table 21-99. PCI Express Uncorrectable Error Status Register Field Description (continued)

Bits	Name	Description
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.

21.3.10.3 PCI Express Uncorrectable Error Mask Register—0x108

The PCI Express uncorrectable error mask register is shown in Figure 21-105.

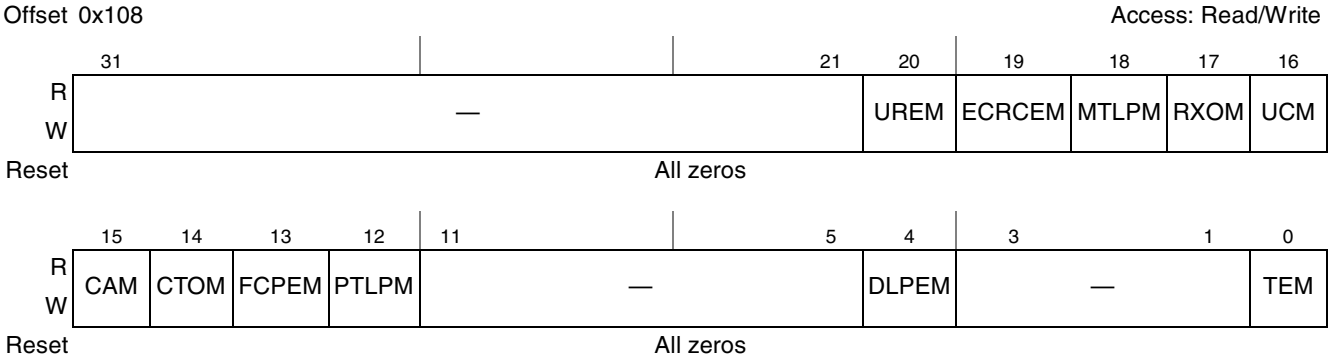


Figure 21-105. PCI Express Uncorrectable Error Mask Register

Table 21-100. PCI Express Uncorrectable Error Mask Register Field Description

Bits	Name	Description
31–21	—	Reserved
20	UREM	Unsupported request error mask.
19	ECRCEM	ECRC error mask.
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.

Table 21-100. PCI Express Uncorrectable Error Mask Register Field Description (continued)

Bits	Name	Description
3–1	—	Reserved
0	TEM	Training error mask.

21.3.10.4 PCI Express Uncorrectable Error Severity Register—0x10C

The PCI Express uncorrectable error severity register is shown in Figure 21-106.

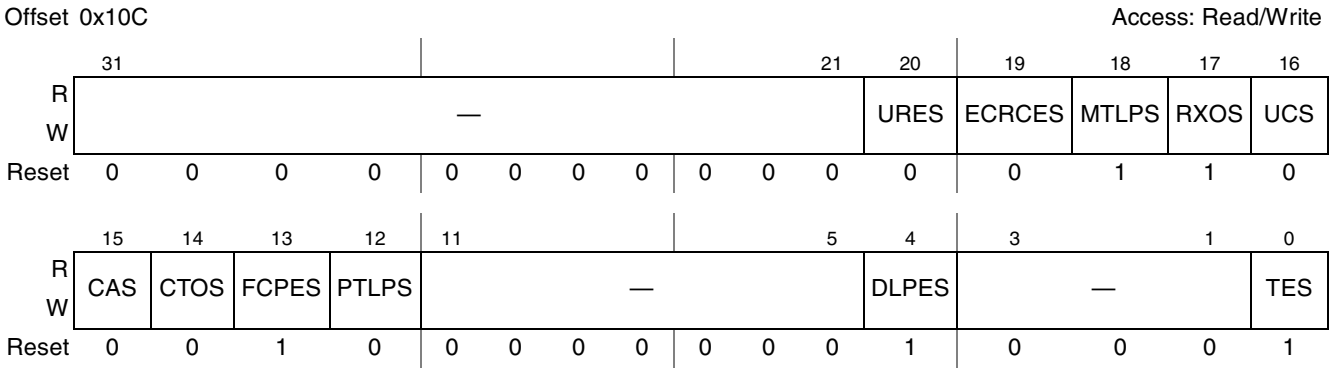


Figure 21-106. PCI Express Uncorrectable Error Severity Register

Table 21-101. PCI Express Uncorrectable Error Severity Register Field 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.
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.

21.3.10.5 PCI Express Correctable Error Status Register—0x110

The PCI Express correctable error status register is shown in Figure 21-107.

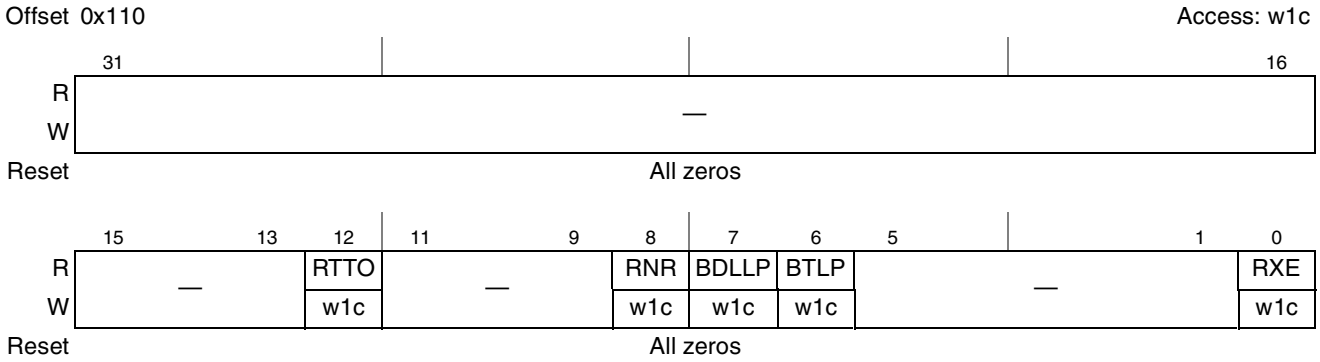


Figure 21-107. PCI Express Correctable Error Status Register

Table 21-102. PCI Express Correctable Error Status Register Field Description

Bits	Name	Description
31–13	—	Reserved
12	RTTO	Replay timer timeout 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

21.3.10.6 PCI Express Correctable Error Mask Register—0x114

The PCI Express correctable error mask register is shown in Figure 21-108.

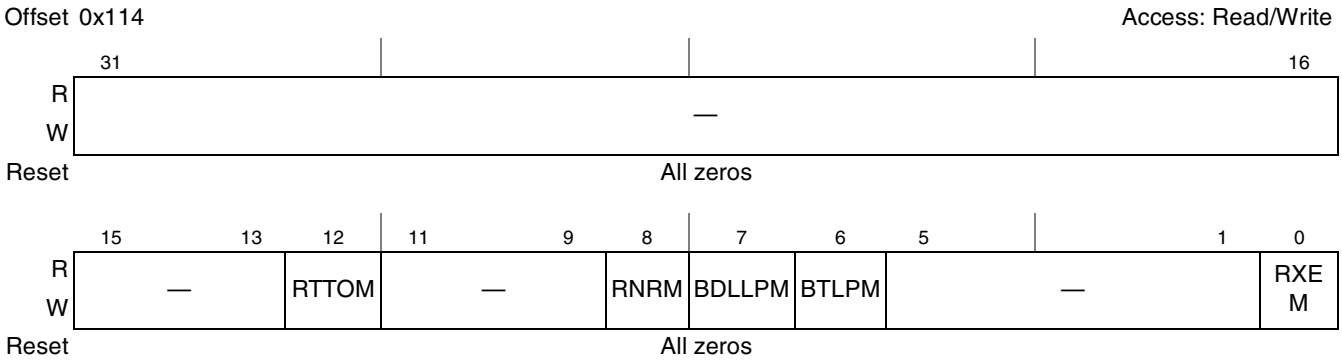


Figure 21-108. PCI Express Correctable Error Mask Register

Table 21-103. PCI Express Correctable Error Mask Register Field 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

21.3.10.7 PCI Express Advanced Error Capabilities and Control Register—0x118

The PCI Express advanced error capabilities and control register is shown in Figure 21-109.

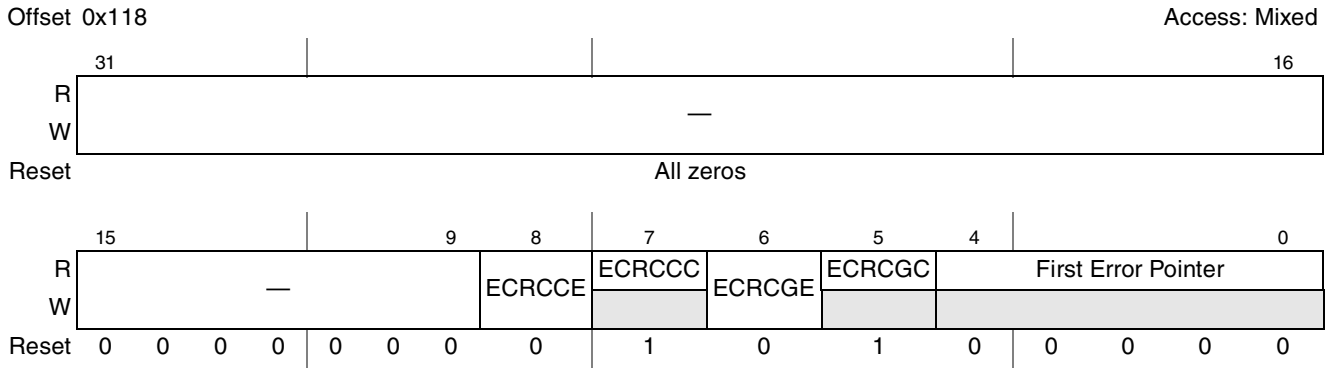


Figure 21-109. PCI Express Advanced Error Capabilities and Control Register

Table 21-104. PCI Express Advanced Error Capabilities and Control Register Field Description

Bits	Name	Description
31–9	—	Reserved.
8	ECRCCE	ECRC checking enable.
7	ECRCCC	ECRC checking capable.
6	ECRCGE	ECRC generation enable.
5	ECRCGC	ECRC generation capable.
4–0	First Error Pointer	The First Error Pointer is a read-only register that identifies the bit position of the first error reported in the Uncorrectable Error Status register.

21.3.10.8 PCI Express Header Log Register—0x11C–0x12B

The PCI Express header log register is shown in Figure 21-110.

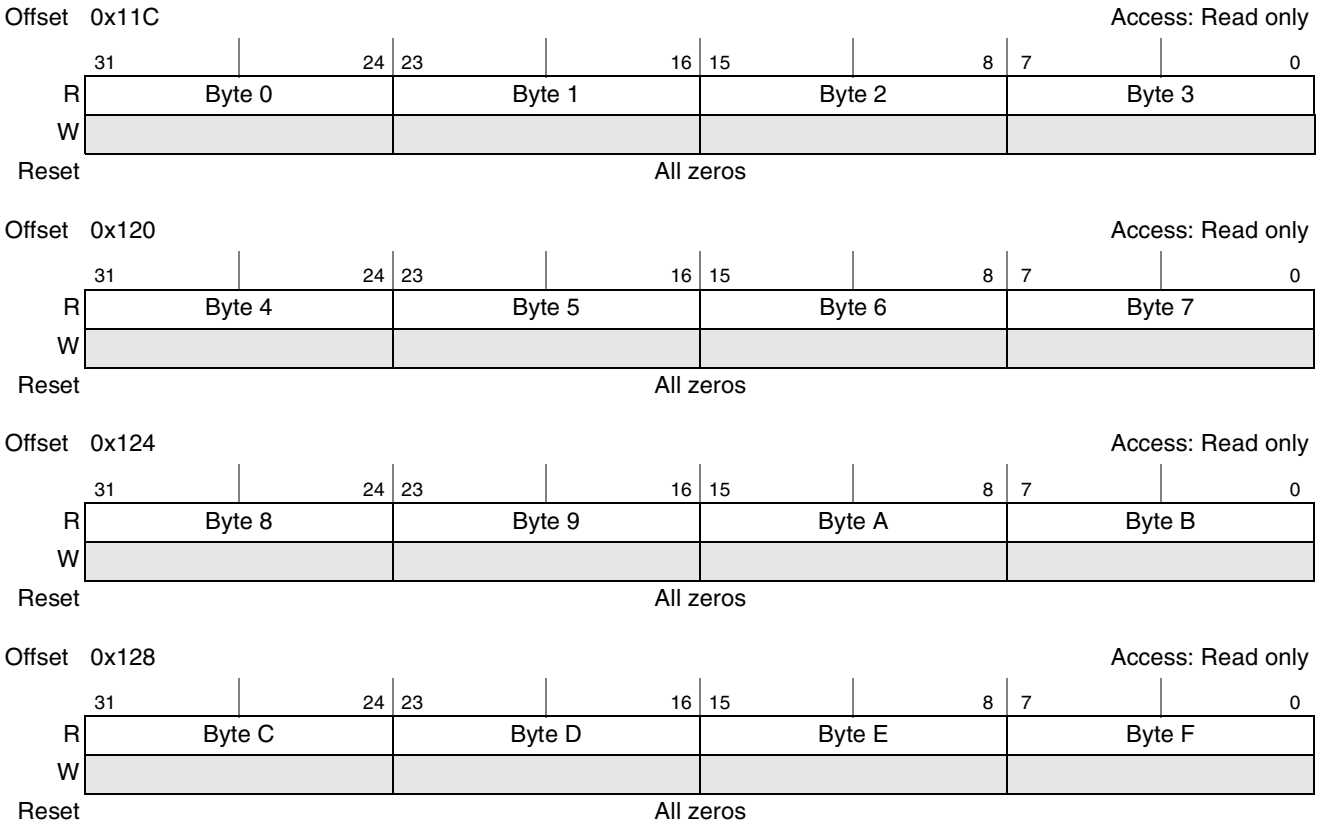


Figure 21-110. PCI Express Header Log Register

Table 21-105. PCI Express Header Log Register Field Description

Bits	Name	Description
127–0	Header Log	Header of TLP associated with error.

21.3.10.9 PCI Express Root Error Command Register—0x12C

The PCI Express root error command register is shown in Figure 21-111.

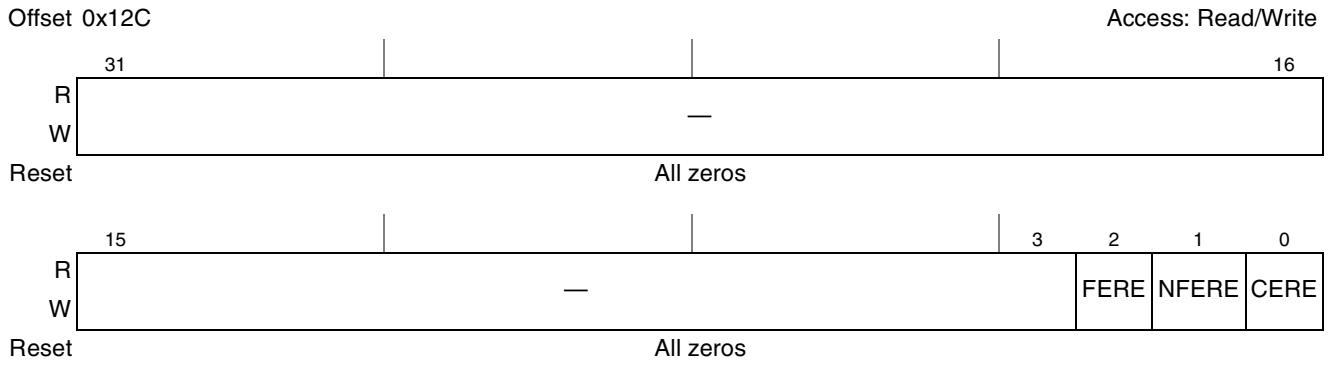


Figure 21-111. PCI Express Root Error Command Register

Table 21-106. PCI Express Root Error Command Register Field 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

21.3.10.10 PCI Express Root Error Status Register—0x130

The PCI Express root error status register is shown in Figure 21-112.

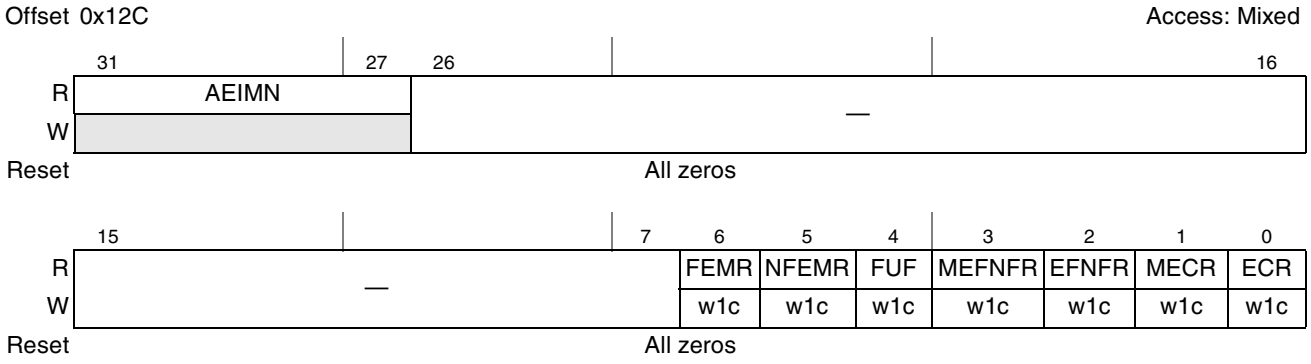


Figure 21-112. PCI Express Root Error Status Register

Table 21-107. PCI Express Root Error Command Status Field Description

Bits	Name	Description
31–27	AEIMN	Advanced error interrupt message number.
26–7	—	Reserved
6	FEMR	Fatal error messages received.

Table 21-107. PCI Express Root Error Command Status Field Description (continued)

Bits	Name	Description
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.

21.3.10.11 PCI Express Correctable Error Source ID Register—0x134

The PCI Express correctable error source ID register is shown in Figure 21-113.

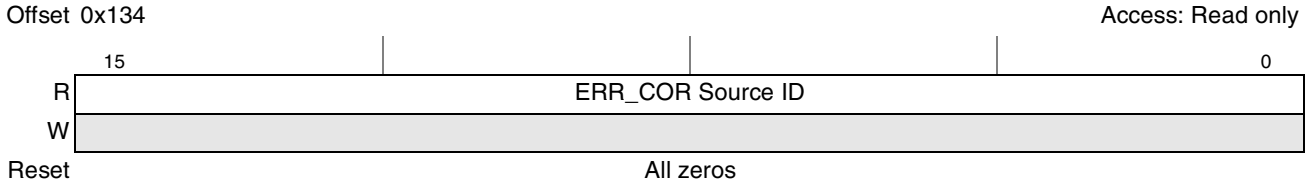


Figure 21-113. PCI Express Correctable Error Source ID Register

Table 21-108. PCI Express Correctable Error Source ID Register Field Description

Bits	Name	Description
15–0	ERR_COR Source ID	Loaded with the Requestor ID indicated in the received ERR_COR Message when the ERR_COR Received register is not already set. Default value of this field is 0.

21.3.10.12 PCI Express Error Source ID Register—0x136

The PCI Express error source ID register is shown in Figure 21-114.

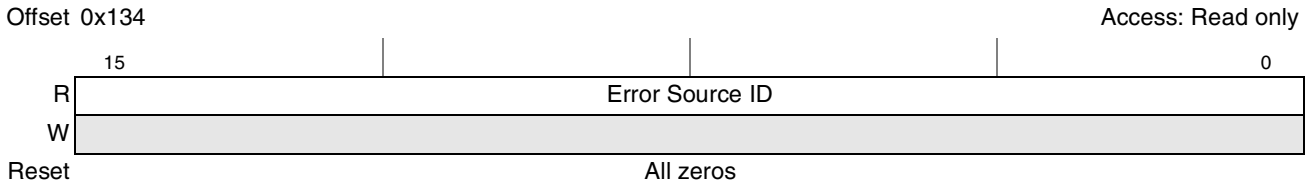


Figure 21-114. PCI Express Correctable Error Source ID Register

Table 21-109. PCI Express Correctable Error Source ID Register Field Description

Bits	Name	Description
15–0	Error Source ID	ERR_FATAL/NONFATAL source ID

21.3.10.13 LTSSM State Status Register—0x404

The PCI Express link training and status state machine (LTSSM) state status register, shown in Figure 21-115, provides detailed information about link training status. This register is useful for debugging link training failures.

NOTE

The Status Code in PEX_LTSSM_STAT changes while reacting to the link status. Therefore, software may need to read PEX_LTSSM_STAT multiple times to ensure the value has stabilized.

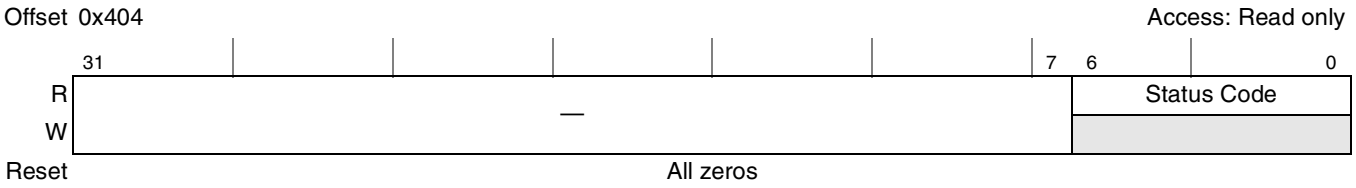


Figure 21-115. PCI Express LTSSM State Status Register (PEX_LTSSM_STAT)

The fields of the PCI Express LTSSM state status register are described in Table 21-110.

Table 21-110. PEX_LTSSM_STAT Field Descriptions

Bits	Name	Description
31–7	—	Reserved
6–0	Status code	Status code. See Table 21-111 for encodings.

Table 21-111 provides the encodings for the status code field of the PEX_LTSSM_STAT register.

Table 21-111. 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)

Table 21-111. PEX_LTSSM_STAT Status Codes (continued)

Status Code (Hex)	LTSSM State Description	Status Code (Hex)	LTSSM State Description
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)
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

21.3.10.14 PCI Express Controller Core Clock Ratio Register—0x440

The PCI Express controller clock frequency is determined by the platform clock frequency and the state of the cfg_platform_freq configuration signal at reset, according to the following expression:

$$\text{PCI Express Controller Clock Frequency} = \frac{\text{Platform Clock Frequency}}{(1 + \text{cfg\_platform\_freq})}$$

Therefore, if the platform clock frequency is less than or equal to 400 MHz, then cfg_platform_freq is 0 and the PCI Express controller clock frequency is the same as the platform clock frequency; if the platform clock frequency is greater than or equal to 500 MHz, then cfg_platform_freq is 1 and the PCI Express controller clock frequency is one-half the platform clock frequency.

The PCI Express controller core clock ratio register, shown in Figure 21-116, is used to program the ratio of the actual PCI Express controller clock frequency to the default controller core frequency (333 MHz). This is required only when a PCI Express controller clock frequency other than the default 333 MHz has to be used.

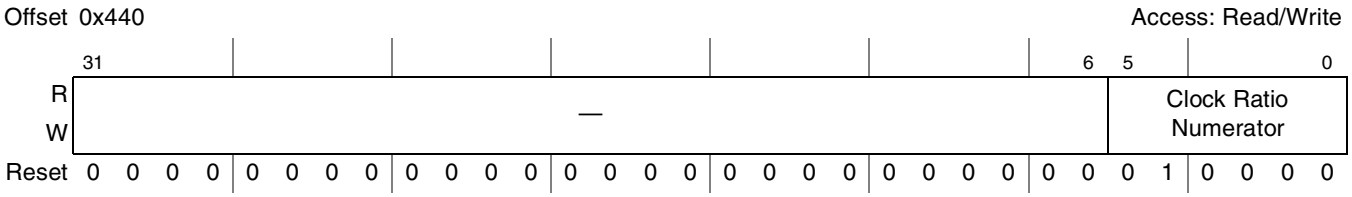


Figure 21-116. PCI Express IP Block Core Clock Ratio Register (PEX_GCLK_RATIO)

The fields of the PCI Express IP block core clock ratio register are described in Table 21-112.

Table 21-112. PEX_GCLK_RATIO Field Descriptions

Bits	Name	Description
31–6	—	Reserved
5–0	Clock Ratio Numerator	The numerator of the ratio of the actual PCI Express controller clock frequency used to the default core clock frequency of 333 MHz. 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)

As an example of programming PEX_GCLK_RATIO, consider the case where the actual PCI Express controller clock is 250 MHz, the ratio of the actual clock to the default clock (333 MHz) is 3:4. that is, the default core clock has to be multiplied by the ratio (3/4, which is equivalent to 12/16). So the register has to be programmed with the decimal numerator value 12 or 0x0000_000C.

21.3.10.15 PCI Express Power Management Timer Register—0x450

The PCI Express power management timer register, shown in Figure 21-117, is used to program the time-in values for entering L0s and L1 power management states.

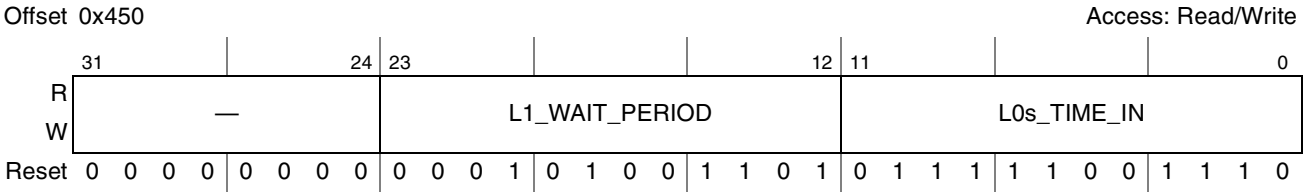


Figure 21-117. PCI Express Power Management Timer Register (PEX_PM_TIMER)

The fields of the PCI Express power management timer register are described in Table 21-113.

Table 21-113. PEX_PM_TIMER Field Descriptions

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 333 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 333 MHz.

21.3.10.16 PCI Express PME Time-Out Register (EP-Mode Only)—0x454

The PCI Express PME time-out register, shown in Figure 21-118, 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.

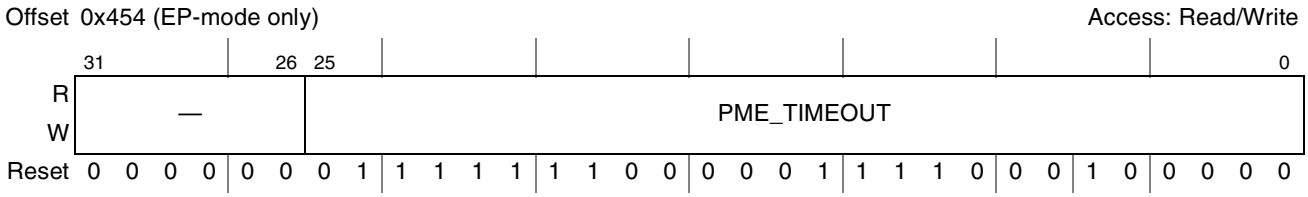


Figure 21-118. PCI Express PME Time-Out Register (PEX_PME_TIMEOUT)

The fields of the PCI Express PME time-out register are described in Table 21-113.

Table 21-114. PEX_PME_TIMEOUT Field Descriptions

Bits	Name	Description
31–26	—	Reserved
25–0	PME_TIMEOUT	The PME time-out value specifies the interval before PME messages are resent by the controller, provided the PME_STAT bit in the PCI Express power management status and control register (offset 0x48) 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 333 MHz.

21.3.10.17 PCI Express Subsystem Vendor ID Update Register (EP Mode Only)—0x478

The PCI Express subsystem vendor ID update register, shown in Figure 21-119, is used to set the values for the Subsystem ID and Subsystem Vendor ID registers in the Type 0 configuration header.

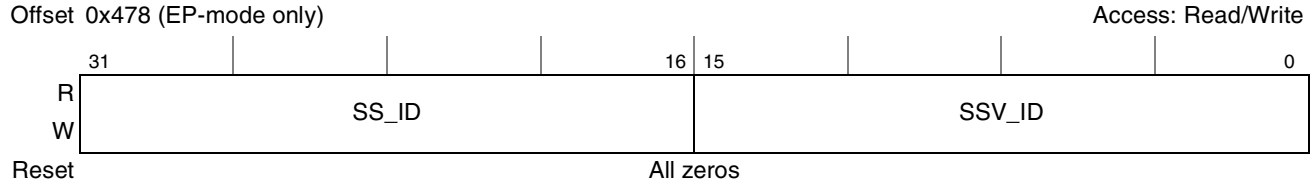


Figure 21-119. PCI Express Subsystem Vendor ID Update Register (PEX_SSVID_UPDATE)

The fields of the PCI Express subsystem vendor ID update register are described in Table 21-115.

Table 21-115. PEX_SSVID_UPDATE Field Descriptions

Bits	Name	Description
31–16	SS_ID	Subsystem ID [15–0] value
15–0	SSV_ID	Subsystem vendor ID [15–0] value

When used as an endpoint, the controller's initialization software programs the desired subsystem ID and subsystem vendor ID values in PEX_SSVID_UPDATE before setting the CFG_READY bit in the PEX_CFG_READY register (see Section 21.3.10.18, "Configuration Ready Register—0x4B0"). That way, when the host begins system enumeration, the correct values are present in the Type 0 configuration header.

21.3.10.18 Configuration Ready Register—0x4B0

The PCI Express configuration ready register, shown in Figure 21-120, is used to indicate 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). The CFG_READY bit in this register should be set after all relevant configuration registers have been programmed. This makes sure the external host reads the correct capability advertisements during enumeration.

Note that the state of PEX_CFG_READY[CFG_READY] is dependent upon the POR configuration setting described in Section 21.5.1, "Boot Mode and Inbound Configuration Transactions."

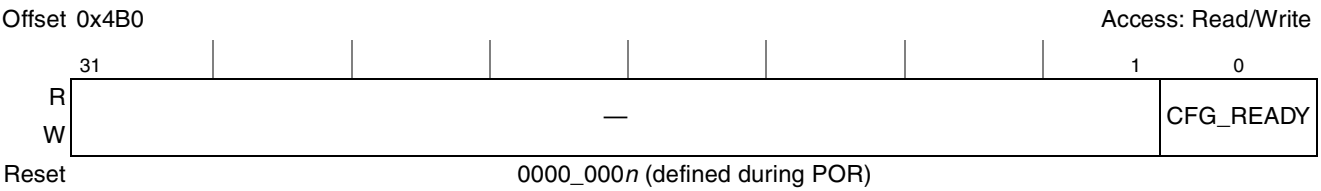


Figure 21-120. PCI Express Configuration Ready Register (PEX_CFG_READY)

The fields of the PCI Express configuration ready register are described in Table 21-116.

Table 21-116. PEX_CFG_READY Field Descriptions

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) Note that the reset state of this bit is determined during POR.

21.3.10.19 Flow Control Update Timeout Register—0x4B8

The PCI Express flow control update timeout register, shown in Figure 21-121, is used to program the timeout value for update-FC DLLP reception in terms of PCI Express controller core clock cycles.

Offset 0x4B8 Access: Read/Write

	31	30														18	17														0
R W	DIS_LR	—															FC_UPDATE_TIMEOUT														
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0

Figure 21-121. PCI Express Flow Control Update Timeout Register (PEX_FC_UPDATE_TOR)

The fields of the PCI Express flow control update timeout register are described in Table 21-117.

Table 21-117. PEX_FC_UPDATE_TOR Field Descriptions

Bits	Name	Description
31	DIS_LR	Disable link retraining on FC update timeout error. The PCI Express protocol allows for optional link retraining by a device if FC updates are not received from the remote device within the timeout period programmed in bits [17–0] of this register. If the FC updates were lost due to link error issues, this retraining will help to restore normal operation. By default, this retraining is disabled. Since this is an optional feature, the user is allowed to enable or disable such automatic retraining, if desired. 1 Disable link retraining on FC update timeout error. 0 Enable link retraining on FC update timeout error.
30–18	—	Reserved
17–0	FC_UPDATE_TIMEOUT	FC update timeout value specifies the interval (in PCI Express controller core clock cycles) to wait for the reception of consecutive FC-update DLLPs before indicating a flow-control protocol error. The value is calculated as: Time (in µsec) × PCI Express controller core clock frequency (in MHz) The maximum time value is 200 µsec; the default value (0x10428) is 200 µsec for the default clock frequency of 333 MHz.

21.3.10.20 Secondary Status Interrupt Mask Register (RC-Mode Only)—0x5A0

The PCI Express secondary status interrupt mask register, shown in Figure 21-122, can be used to disable sideband interrupt generation when error bits in the PCI Express secondary status register are set. See Section 21.3.8.3.8, “PCI Express Secondary Status Register—0x1E,” for more information. By default, interrupt generation due to secondary status errors is disabled.

Offset 0x5A0 (RC-mode only) Access: Read/write

	31–6	5	4	3	2	1	0
R W	—	M_DPE	M_SSE	M_RMA	M_RTA	M_STA	M_MDPE
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 21-122. PCI Express PCI Interrupt Mask Register (PEX_SS_INTR_MASK)

The fields of the PCI Express secondary status interrupt mask register are described in Table 21-118.

Table 21-118. PEX_SS_INTR_MASK Field Descriptions

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

21.4 Functional Description

The PCI Express protocol relies on a requestor/completer relationship where one device requests that some desired action be performed by some target device 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.

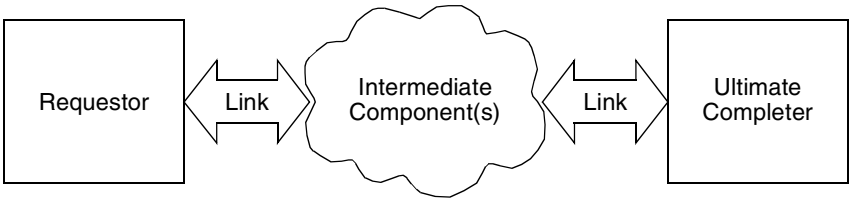


Figure 21-123. Requestor/Completer Relationship

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 21-124.

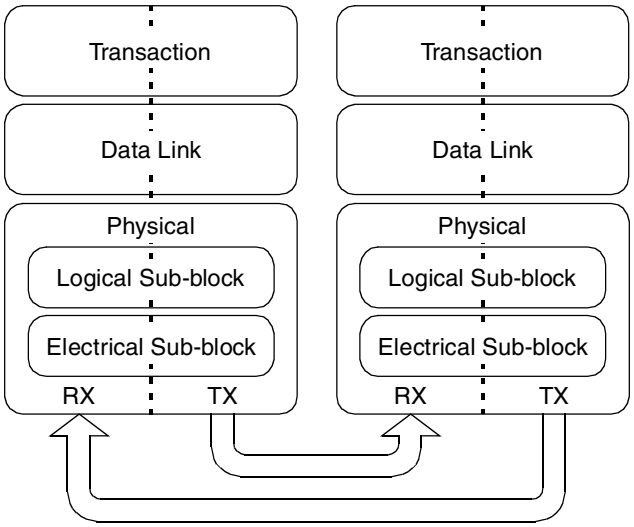


Figure 21-124. PCI Express High-Level Layering

Packets are formed in the transaction layer (TLPs) and data link layer (DLLPs), and each subsequent layer adds the necessary encodings and framing—as shown in Figure 21-125. 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.

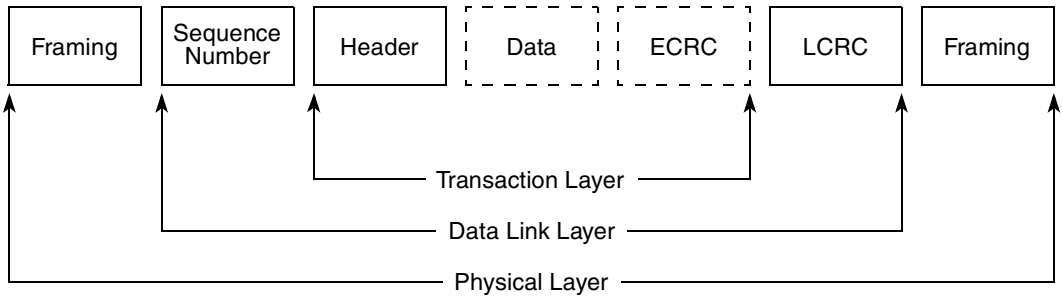


Figure 21-125. PCI Express Packet Flow

21.4.1 Architecture

This section describes implementation details of the PCI Express controller.

21.4.1.1 PCI Express Transactions

Table 21-119 contains the list of transactions that the PCI Express controller supports as an initiator and a target.

Table 21-119. PCI Express Transactions

PCI Express Transaction	Supported as an Initiator	Supported as a Target	Definition
Mrd	Yes	Yes	Memory Read Request
MRdLk	No	No	Memory Read Lock. As a target, CplLk with UR status is returned.
MWr	Yes	Yes	Memory Write Request to memory-mapped PCI-Express space
IORd	Yes (RC only)	No	I/O Read request. As a target, Cpl with UR status is returned.
IOWr	Yes (RC only)	No	I/O Write Request. As a target, Cpl with UR status is returned.
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. As a target, Cpl with UR status is returned.
CfgWr1	Yes (RC only)	No	Configuration Write Type 1. As a target, Cpl with UR status is returned.
Msg	Yes	Yes	Message Request
MsgD	Yes (RC only)	Yes (EP only)	Message Request with Data payload. Note that Set_Slot_Power_Limit is the only message with data that is supported and then only when the controller is an initiator and in RC mode or a target and in EP mode.
Cpl	Yes	Yes	Completion without Data
CplD	Yes	Yes	Completion with Data
CplLk	No	Yes	Completion for Locked Memory Read without Data. The only time that CplLk is returned with UR status is when the controller receives a MRdLk command.
CplDLk	No	No	Completion for Locked Memory Read with Data

21.4.1.2 Byte Ordering

Whenever data must cross a bridge between two busses, the byte ordering of data on the source and destination buses must be considered. The internal platform bus of this device is inherently big endian and the PCI Express bus interface is inherently little endian.

There are two methods to handle ordering of data as it crosses a bridge—address invariance and data invariance. Address invariance preserves the addressing of bytes within a scalar data element, but not the relative significance of the bytes within that scalar. Conversely, data invariance preserves the relative significance of bytes within a scalar, but not the addressing of the individual bytes that make up a scalar.

This device uses address invariance as its byte ordering policy.

As stated above, address invariance preserves the byte address of each byte on an I/O interface as it is placed in memory or moved into a register. This policy can have the effect of reversing the significance order of bytes (most significant to least significant and vice versa), but it has the benefit of preserving the

format of general data structures. Provided that software is aware of the endianness and format of the data structure, it can correctly interpret the data on either side of the bridge.

Figure 21-126 shows the transfer of a 4-byte scalar, 0x4142_4344, from a big endian source across an address invariant bridge to a little endian destination.

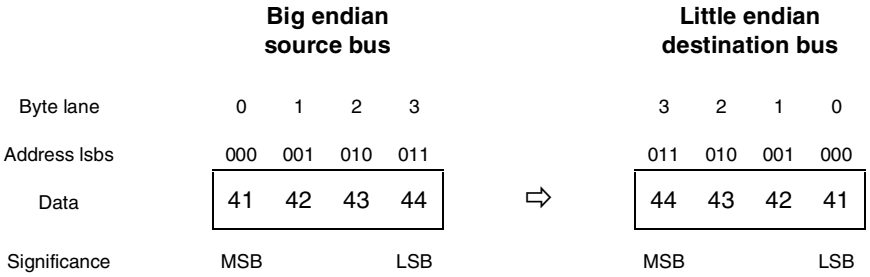


Figure 21-126. Address Invariant Byte Ordering—4 bytes Outbound

Note that although the significance of the bytes within the scalar have changed, the address of the individual bytes that make up the scalar have not changed. As long as software is aware that the source of the data used a big endian format, the data can be interpreted correctly.

Figure 21-128 shows data flowing the other way, from a little endian source to a big endian destination.

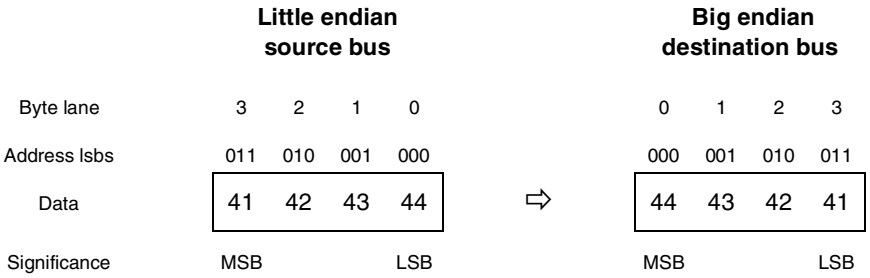


Figure 21-127. Address Invariant Byte Ordering—4 bytes Inbound

Figure 21-128 shows an outbound transfer of an 8-byte scalar, 0x5455_1617_CDCE_2728, using address invariance.

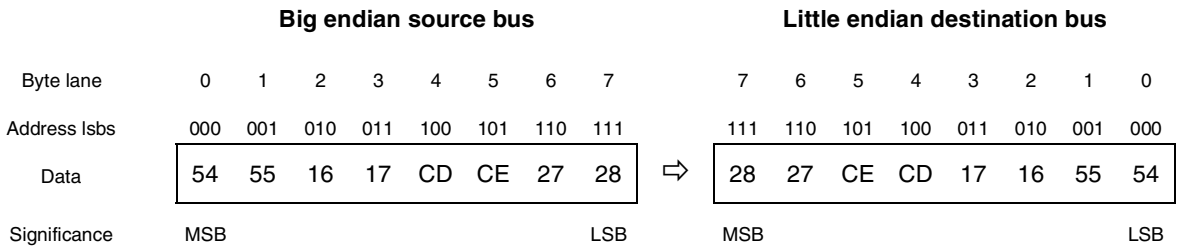


Figure 21-128. Address Invariant Byte Ordering—8 bytes Outbound

Figure 21-129 shows an inbound transfer of a 2-byte scalar, 0x5837, using address invariance.

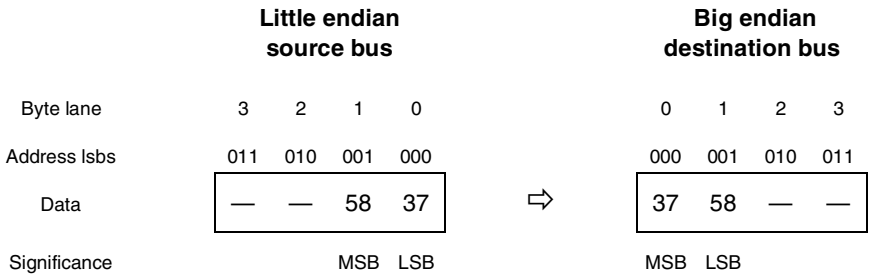


Figure 21-129. Address Invariant Byte Ordering—2 bytes Inbound

Note that in all of these examples, the original addresses of the individual bytes within the scalars (as created by the source) have been preserved.

21.4.1.2.1 Byte Order for Configuration Transactions

All internal memory-mapped registers in the CCSR space use big endian byte ordering. However, the PCI Express specification defines PCI Express configuration registers as little endian. All accesses to the PCI Express configuration port, PEX_CONFIG_DATA, including the those targeting the internal PCI Express configuration registers, use the address invariance policy as shown in Figure 21-130. Therefore, software must access PEX_CONFIG_DATA with little-endian formatted data—either using the **lwbrx**/**stwbrx** instructions or by manipulating the data before writing to and after reading from PEX_CONFIG_DATA.

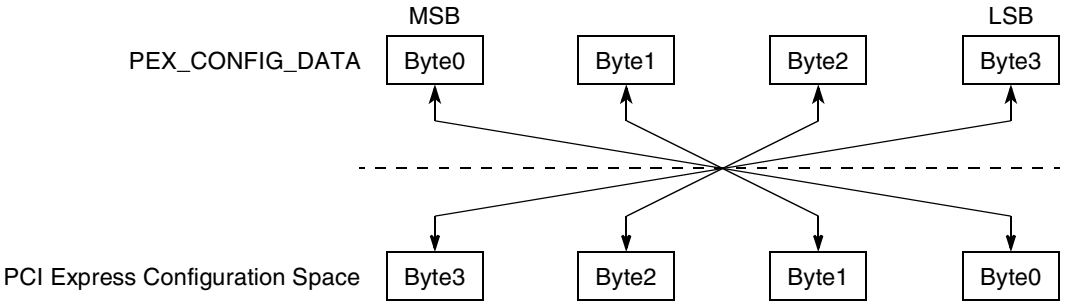


Figure 21-130. PEX_CONFIG_DATA Byte Ordering

21.4.1.3 Lane Reversal

The PCI Express link supports lane reversal. Table 21-120 describes the supported configurations.

Table 21-120. Lane Assignment With and Without Lane Reversal

Link Configuration	Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
x8 link without lane reversal	0	1	2	3	4	5	6	7
x4 link without lane reversal	0	1	2	3	—	—	—	—
x2 link without lane reversal	0	1	—	—	—	—	—	—
x1 link without lane reversal	0	—	—	—	—	—	—	—
x8 link with lane reversal	7	6	5	4	3	2	1	0

Table 21-120. Lane Assignment With and Without Lane Reversal (continued)

Link Configuration	Lane 0	Lane 1	Lane 2	Lane 3	Lane 4	Lane 5	Lane 6	Lane 7
x4 link with lane reversal	—	—	—	—	3	2	1	0
x2 link with lane reversal	—	—	—	—	—	—	1	0
x1 link with lane reversal	—	—	—	—	—	—	—	0

Note: The numbers shown in this table (0–7) are the lane numbers assigned to each lane as a result of link initialization and configuration.
— indicates that the lane is not part of the configured link.

Note that lane reversal is only effective for devices that use the full 8 lanes. That is, if a x4 device is connected to lanes 0–3 and the link training fails without lane reversal, the lane reversal causes the link to attempt connection on lanes 7–4 which would be impossible.

21.4.1.4 Transaction Ordering Rules

In general, transactions are serviced in the order that they are received. However, transactions can be reordered as they are sent due to a stalled condition such as a full internal buffer. The following are the ordering rules for sending the next outstanding request:

- A posted request can and bypasses all other transactions except another posted request.
- A completion can and only bypasses non-posted. It can and bypasses posted requests only if the relaxed ordering (RO) bit is set.
- A non-posted request cannot bypass posted or other non-posted requests, but it can bypass a completion if the relaxed ordering (RO) bit is set.

21.4.1.5 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 TLP header for a 32-bit address packet is 0; a 64-bit address packet has a FMT[0] = 1. The PCI Express TLP header for a memory read transaction has TYPE[4:0] = 00000 and FMT[1] = 0. A memory write transaction has TYPE[4:0] = 00000 and FMT[1] = 1. As an initiator, the controller is capable of sending 32- or 64-bit memory packets. Any transaction from the internal platform that (after passing through the translation mechanism) has a translated address greater than 4G is sent as a 64-bit memory packet. Otherwise, a 32-bit memory packet is sent. As a target device, the controller is capable of decoding 32- or 64-bit memory packets. This is done through two 32-bit inbound windows and two 64-bit inbound windows. All inbound addresses are translated to 36-bit internal platform addresses.

21.4.1.6 I/O Space Addressing

The controller does not support I/O transactions as a target. As an initiator, the controller can send I/O transactions in RC mode only. This can be done by programming one of the outbound translation window's attribute to send I/O transactions. All I/O transactions only access 32-bit address I/O space. The PCI Express TLP header for an I/O read transaction has TYPE[4:0] = 00010 and FMT[1] = 0. The PCI Express TLP header for an I/O write transaction has TYPE[4:0] = 00010 and FMT[1] = 1.

21.4.1.7 Configuration Space Addressing

As an initiator, the controller supports both type 0 and type 1 configuration cycles when configured in RC mode. There are two methods of generating a configuration transaction; refer to Section 21.3.7, "PCI Express Configuration Space Access," for more information. A configuration transaction can hit into the controller's internal configuration space, it can be sent out on the PCI Express link, or it can be internally terminated. The PCI Express TLP header for a type 0 configuration read transaction has TYPE[4:0] = 00100 and FMT[1] = 0; the PCI Express TLP header for a type 0 configuration write transaction has TYPE[4:0] = 00100 and FMT[1] = 1. The PCI Express TLP header for a type 1 configuration read transaction has TYPE[4:0] = 00101 and FMT[1] = 0; the PCI Express TLP header for a type 1 configuration write transaction has TYPE[4:0] = 00101 and FMT[1] = 1. Note that all configuration transactions sent on PCI Express require a response regardless whether they are read or a write configuration transactions.

The controller does not generate configuration transactions in EP mode. Only inbound configuration transactions are supported in EP mode.

21.4.1.8 Serialization of Configuration and I/O Writes

Configuration and I/O writes are serialized by the controller. The logic after issuing a configuration write or IO write does not issue any new transactions until the outstanding configuration or I/O write is finished. This means that an acknowledgement packet from the link partner in the form of a CpL TLP packet must be seen or the transaction has timed out. If the CpL packet contains a CRS status, then the logic re-issues the configuration write transaction. It keeps retrying the request until either a status other than CRS is returned or the transaction times out.

Note that it is possible for outbound configuration read request to be requeued and be placed at the end of the request queue due to CRS condition.

21.4.1.9 Messages

Software message generation is supported in both RC and EP modes.

21.4.1.9.1 Outbound ATMU Message Generation

Software can choose to send a message by programming PEXOWAR*n*[WTT] = 0x5. A message is sent by writing a 4-byte transaction in big-endian format that hits in an outbound window configured to send messages.

Part of the 4-byte data is used to store information such as message code and routing information. Table 21-121 describes the message data format.

Table 21-121. Internal Platform (OCeaN) Message Data Format

Bits	Name	Reset Value	Description
0–15	—	—	Reserved
16–18	Routing	x	Routing mechanism. Contains the message's routing information

Table 21-121. Internal Platform (OCeaN) Message Data Format

Bits	Name	Reset Value	Description
19–23	—	—	Reserved
24–31	Code	x	Message code. Contains the actual message type to be sent.

In addition to the outbound ATMU, the PEX PM Command register also provides the capability to send PME_Turn_Off message or PM_PME message by setting bits 31 or 29. See Section 21.3.3.4, “PCI Express Power Management Command Register (PEX_PMCR),” on page 21-18 for more information.

Table 21-122 provides a complete list of supported outbound messages depending on whether RC or EP is configured.

Table 21-122. PCI Express ATMU Outbound Messages

Name	Code[7:0]	Routing[2:0]	RC	EP	Description
Assert_INTA	0010 0000	100	N/A	Yes	Sent upstream by endpoint
Assert_INTB	0010 0001	100	N/A	Yes	Sent upstream by endpoint
Assert_INTC	0010 0010	100	N/A	Yes	Sent upstream by endpoint
Assert_INTD	0010 0011	100	N/A	Yes	Sent upstream by endpoint
Deassert_INTA	0010 0100	100	N/A	Yes	Sent upstream by endpoint
Deassert_INTB	0010 0101	100	N/A	Yes	Sent upstream by endpoint
Deassert_INTC	0010 0110	100	N/A	Yes	Sent upstream by endpoint
Deassert_INTD	0010 0111	100	N/A	Yes	Sent upstream by endpoint
PM_Active_State_Nak	0001 0100	100	Yes	N/A	Terminate at receiver
PM_PME	0001 1000	000	N/A	Yes	Sent Upstream by PME-requesting component
PME_Turn_Off	0001 1001	011	Yes	N/A	Broadcast Downstream
PM_TO_Ack	0001 1011	101	N/A	Yes	Sent Upstream by Endpoint
ERR_COR	0011 0000	000	N/A	Yes	Sent by component when it detects a correctable error
ERR_NONFATAL	0011 0001	000	N/A	Yes	Sent by component when it detects a Non-fatal, uncorrectable error
ERR_FATAL	0011 0011	000	N/A	Yes	Sent by component when it detects a Fatal, uncorrectable error
Unlock	0000 0000	000	No	N/A	Not supported
Set_Slot_Power_Limit	0101 0000	100	Yes	N/A	Set Slot Power Limit in Upstream Port
Vendor_Defined Type 0	0111 1110		No	No	Not supported
Vendor_Defined Type 1	0111 1111		No	No	Not supported
Attention_Indicator_On	0100 0001	100	Yes	N/A	Hot-plug message

Table 21-122. PCI Express ATMU Outbound Messages (continued)

Name	Code[7:0]	Routing[2:0]	RC	EP	Description
Attention_Indicator_Blink	0100 0011	100	Yes	N/A	Hot-plug message
Attention_Indicator_Off	0100 0000	100	Yes	N/A	Hot-plug message
Power_Indicator_On	0100 0101	100	Yes	N/A	Hot-plug message
Power_Indicator_Blink	0100 0111	100	Yes	N/A	Hot-plug message
Power_Indicator_Off	0100 0100	100	Yes	N/A	Hot-plug message
Attention_Button_Pressed	0100 1000	100	N/A	Yes	Hot-plug message

21.4.1.9.2 Inbound Messages

Table 21-123 provides a complete list of supported inbound messages in RC mode.

Table 21-123. PCI Express RC Inbound Message Handling

Name	Code[7:0]	Routing[2:0]	Action
Assert_INTA	0010 0000	100	Send to PIC
Assert_INTB	0010 0001	100	Send to PIC
Assert_INTC	0010 0010	100	Send to PIC
Assert_INTD	0010 0011	100	Send to PIC
De-assert_INTA	0010 0100	100	Send to PIC
De-assert_INTB	0010 0101	100	Send to PIC
De-assert_INTC	0010 0110	100	Send to PIC
De-assert_INTD	0010 0111	100	Send to PIC
PM_Active_State_Nak	0001 0100	100	No action taken
PM_PME	0001 1000	000	Generate interrupt to PIC if enabled
PME_Turn_Off	0001 1001	011	No action taken
PME_TO_Ack	0001 1011	101	Set PEX_PME_MES_DR[ENL23] bit and generate interrupt to PIC if enabled
ERR_COR	0011 0000	000	Generate interrupt to PIC if enabled
ERR_NONFATAL	0011 0001	000	Generate interrupt to PIC if enabled
ERR_FATAL	0011 0011	000	Generate interrupt to PIC if enabled
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

Table 21-123. PCI Express RC Inbound Message Handling (continued)

Name	Code[7:0]	Routing[2:0]	Action
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
Power_Indicator_Off	0100 0100	100	No action taken
Attention_Button_Pressed	0100 1000	100	Set PEX_PME_MES_DR[ABP] bit and send interrupt if enabled.

Table 21-124 provides a complete list of supported inbound messages in EP mode.

Table 21-124. 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	Set PEX_PME_MES_DR[PTO] bit. 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	Update power value in PCI Express device capability register in configuration space.
Vendor_Defined Type 0	0111 1110		No action taken
Vendor_Defined Type 1	0111 1111		No action taken
Attention_Indicator_On	0100 0001	100	Set PEX_PME_MES_DR[AION] bit. Send interrupt if enabled.
Attention_Indicator_Blink	0100 0011	100	Set PEX_PME_MES_DR[AIB] bit. Send interrupt if enabled.
Attention_Indicator_Off	0100 0000	100	Set PEX_PME_MES_DR[AIOF] bit. Send interrupt if enabled.
Power_Indicator_On	0100 0101	100	Set PEX_PME_MES_DR[PION] bit. Send interrupt if enabled.

Table 21-124. PCI Express EP Inbound Message Handling (continued)

Name	Code[7:0]	Routing[2:0]	Action
Power_Indicator_Blink	0100 0111	100	Set PEX_PME_MES_DR[PIB] bit. Send interrupt if enabled.
Power_Indicator_Off	0100 0100	100	Set PEX_PME_MES_DR[PIOF] bit. Send interrupt if enabled.
Attention_Button_Pressed	0100 1000	100	No action taken

21.4.1.10 Error Handling

The PCI Express specification classifies errors as correctable and uncorrectable. Correctable errors result in degraded performance, but uncorrectable errors generally result in functional failures. As shown in Figure 21-131 uncorrectable errors can further be classified as fatal or non-fatal.

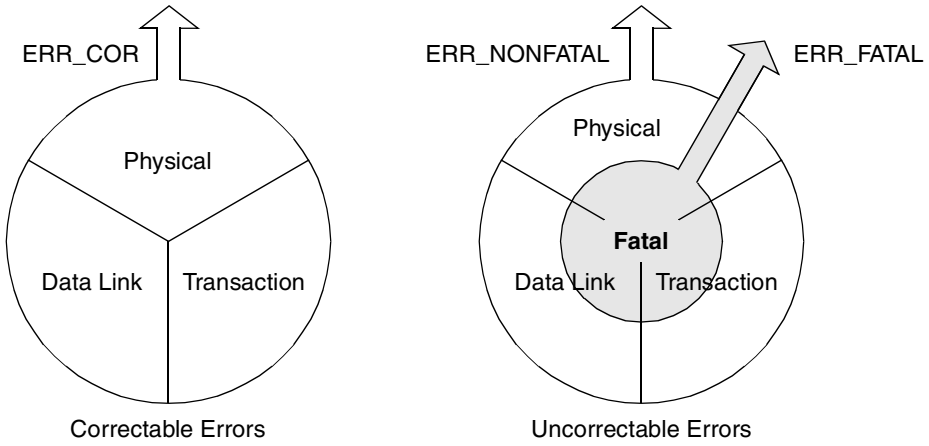


Figure 21-131. PCI Express Error Classification

21.4.1.10.1 PCI Express Error Logging and Signaling

Figure 21-132 shows the PCI Express-defined sequence of operations related to signaling and logging of errors detected by a device. Note that the PCI Express controller on this device supports the advanced error handling capabilities shown within the dotted lines.

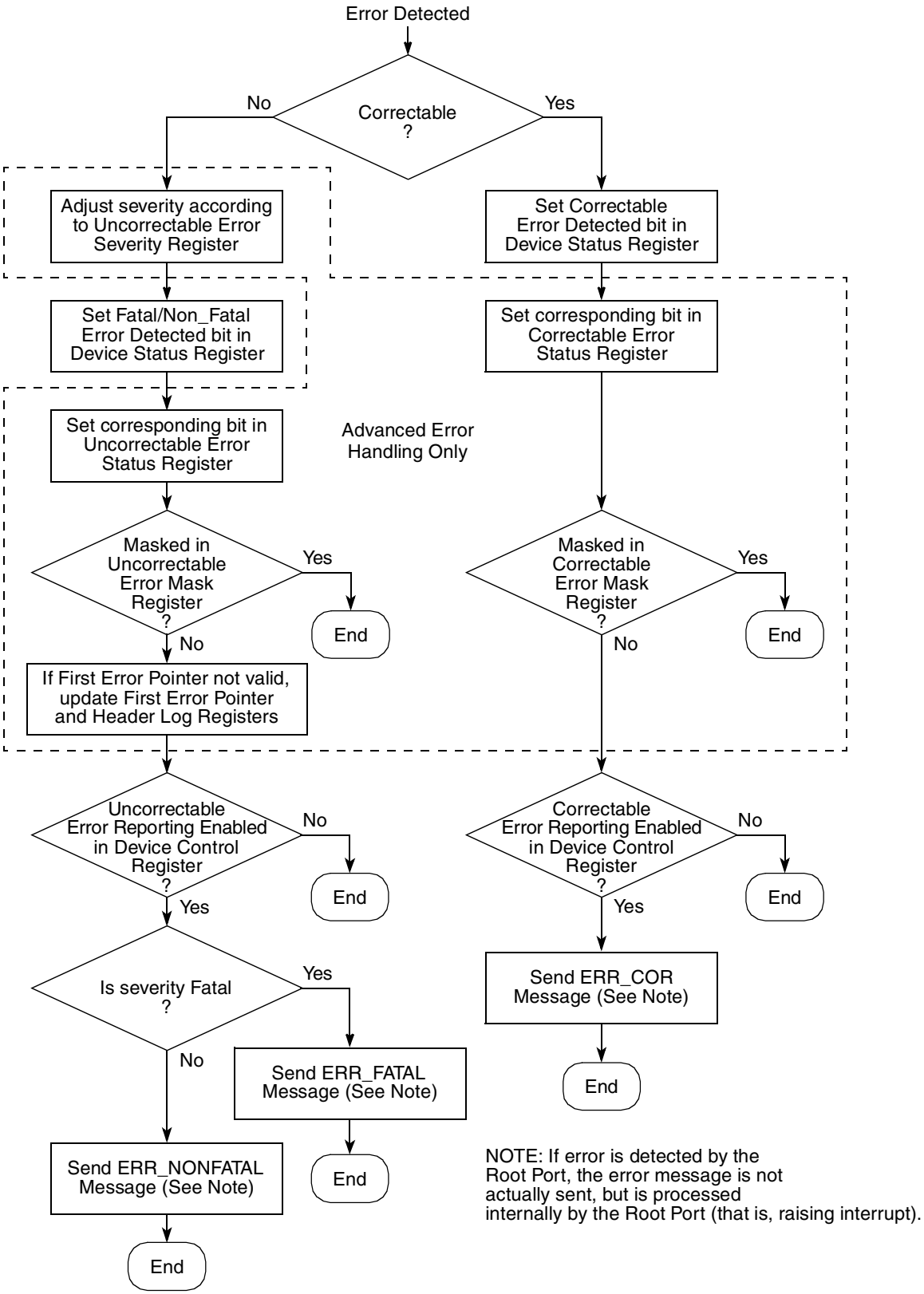


Figure 21-132. PCI Express Device Error Signaling Flowchart

21.4.1.10.2 PCI Express Controller Internal Interrupt Sources

Table 21-125 describes the sources of the PCI Express controller internal interrupt to the PIC and the preconditions for signaling the interrupt.

Table 21-125. PCI Express Internal Controller Interrupt Sources

Status Register Bit	Preconditions
Any bit in PEX_PME_MES_DR set	The corresponding interrupt enable bits must be set in PEX_PME_MES_IER
Any bit in PEX_ERR_DR set	The corresponding interrupt enable bits must be set in PEX_ERR_EN.
PCI Express Root Status Register[16] (PME status) is set	PCI Express Root Control Register [3] (PME interrupt enable) is set
PCI Express Root Error Status Register[6] (fatal error messages received) is set	PCI Express Root Error Command Register [2] (fatal error reporting enable) is set or PCI Express Root Control Register [2] (system error on fatal error enable) is set
PCI Express Root Error Status Register [5] (non-fatal error messages received) is set	PCI Express Root Error Command Register [1] (non-fatal error reporting enable) is set or PCI Express Root Control Register [1] (system error on non-fatal error enable) is set
PCI Express Root Error Status Register[0] (correctable error messages received) is set	PCI Express Root Error Command Register[0] (correctable error reporting enable) is set or PCI Express Root Control Register[0] (system error on correctable error enable) is set.
Any correctable error status bit in PCI Express Correctable Error Status Register is set	The corresponding error mask bit in PCI Express Correctable Error Mask Register is clear and PCI Express Root Error Command Register[0] (correctable error reporting enable) is set
Any fatal uncorrectable error status bit in PCI Express Uncorrectable Error Status Register is set. (The corresponding error is classified as fatal based on the severity setting in PCI Express Uncorrectable Error Severity Register.)	The corresponding error mask bit in PCI Express Uncorrectable Error Mask Register is clear and either PCI Express Device Control Register[2] (fatal error reporting) is set or PCI Express Command Register[8] (SERR) is set.
Any non-fatal uncorrectable error status bit in PCI Express Uncorrectable Error Status Register is set. (The corresponding error is classified as non-fatal based on the severity setting in PCI Express Uncorrectable Error Severity Register.)	The corresponding error mask bit in PCI Express Uncorrectable Error Mask Register is clear and either PCI Express Device Control Register[1] (non-fatal error reporting) is set or PCI Express Command Register[8] (SERR) is set.
PCI Express Secondary Status Register[8] (master data parity error) is set.	PCI Express Secondary Status Interrupt Mask Register[0] (mask master data parity error) is cleared and PCI Express Command Register[6] (parity error response) is set.
PCI Express Secondary Status Register[11] (signaled target abort) is set	PCI Express Secondary Status Interrupt Mask Register[1] (mask signaled target abort) is cleared.
PCI Express Secondary Status Register[12] (received target abort) is set	PCI Express Secondary Status Interrupt Mask Register[2] (mask received target abort) is cleared.

Table 21-125. PCI Express Internal Controller Interrupt Sources (continued)

Status Register Bit	Preconditions
PCI Express Secondary Status Register[13] (received master abort) is set	PCI Express Secondary Status Interrupt Mask Register[3] (mask received master abort) is cleared.
PCI Express Secondary Status Register[14] (signaled system error) is set.	PCI Express Secondary Status Interrupt Mask Register[4] (mask signaled system error) is cleared.
PCI Express Secondary Status Register[15] (detected parity error) is set	PCI Express Secondary Status Interrupt Mask Register[5] (mask detected parity error) is cleared.
PCI Express Slot Status Register[0] (attention button pressed) is set	PCI Express Slot Control Register[0] (attention button pressed enable) is set and PCI Express Slot Control Register[5] (hot plug interrupt enable) is set and either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set.
PCI Express Slot Status Register[1] (power fault detected) is set	PCI Express Slot Control Register[1] (power fault detected enable) is set and PCI Express Slot Control Register[5] (hot plug interrupt enable) is set and either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set.
PCI Express Slot Status Register[2] (MRL sensor changed) is set	PCI Express Slot Control Register[2] (MRL sensor changed enable) is set and PCI Express Slot Control Register[5] (hot plug interrupt enable) is set and either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set.
PCI Express Slot Status Register[3] (presence detect changed) is set	PCI Express Slot Control Register[3] (presence detect changed enable) is set and PCI Express Slot Control Register[5] (hot plug interrupt enable) is set and either PCI Express PM Control Register[1–0] = 00 (the function power state is D0) or PCI Express PM Control Register[8] (PME enable) is set.
PCI Express Slot Status Register[4] (command completed) is set	PCI Express Slot Control Register[4] (command completed interrupt enable) is set and PCI Express Slot Control Register[5] (hot plug interrupt enable) is set.

21.4.1.10.3 Error Conditions

Table 21-126 describes specific error types and the action taken for various transaction types.

Table 21-126. Error Conditions

Transaction Type	Error Type	Action
Inbound response	PEX response time out. This case happens when the internal platform sends a non-posted request that did not get a response back after a specific amount of time specified in the outbound completion timeout register (PEX_OTB_CPL_TOR)	Log error (PEX_ERR_DR[PCT]) and send interrupt to PIC, if enabled.
Inbound response	Unexpected PEX response. This can happen if, after the response times out and the internal queue entry is deallocated, the response comes back.	Log unexpected completion error (PCI Express Uncorrectable Status Register[16]) and send interrupt to PIC, if enabled.
Inbound response	Unsupported request (UR) response status	Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request. Log the error (PEX_ERR_DR[CDNSC] and PCI Express Uncorrectable Status Register[20]) and send interrupt to PIC, if enabled.
Inbound response	Completer abort (CA) response status	Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request. Log the error (PEX_ERR_DR[PCAC, CDNSC] and PCI Express Uncorrectable Status Register[15] and send interrupt to PIC, if enabled.
Inbound response	Poisoned TLP (EP=1)	Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request. Log the error (PCI Express Uncorrectable Status Register[12]) and send interrupt to PIC, if enabled.
Inbound response	ECRC error	Depending upon whether the initial internal request was broken up, the error is not sent until all responses come back for all portions of the internal request. Log the error (PCI Express Uncorrectable Status Register[19]) and send interrupt to PIC, if enabled.
Inbound response	Configuration Request Retry Status (CRS) timeout for a configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA	1.The controller always retries the transaction as soon as possible until a status other than CRS is returned. However, if a CRS status is returned after the configuration retry timeout (PEXCONF_RTY_TOR) timer expires, then the controller aborts the transaction and sends all 1s (0xFFFF_FFFF) data back to requester. 2. Log the error (PEX_ERR_DR[PCT]) and send interrupt to the PIC, if enabled.
Inbound response	UR response for configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA	1. Send back all 1s (0xFFFF_FFFF) data. 2. Log the error (PEX_ERR_DR[CDNSC] and PCI Express Uncorrectable Status Register[20]) and send interrupt to PIC, if enabled.

Table 21-126. Error Conditions (continued)

Transaction Type	Error Type	Action
Inbound response	CA response for Configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA	1. Send back all 1s (0xFFFF_FFFF) data. 2. Log the error (PEX_ERR_DR[PCAC, CDNSC] and PCI Express Uncorrectable Status Register[15]) and send interrupt to PIC, if enabled.
Inbound response	Poisoned TLP (EP=1) response for Configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA	1. Send back all 1s (0xFFFF_FFFF) data. 2. Log the error (PCI Express Uncorrectable Status Register[12]) and send interrupt to PIC, if enabled.
Inbound response	ECRC error response for Configuration transaction that originates from PEX_CONFIG_ADDR/ PEX_CONFIG_DATA	1. Send back all 1s (0xFFFF_FFFF) data. 2. Log the error (PCI Express Uncorrectable Status Register[19]) and send interrupt to PIC, if enabled.
Inbound response	Configuration Request Retry Status (CRS) response for Configuration transaction that originates from ATMU	1.The controller always retries the transaction as soon as possible until a status other than CRS is returned. However, if a CRS status is returned after the configuration retry timeout (PEXCONF_RTY_TOR) timer expires, then the controller aborts the transaction. 2. Log the error (PEX_ERR_DR[CRST]) and send interrupt to the PIC, if enabled.
Inbound response	UR response for Configuration transaction that originates from ATMU	Log the error (PEX_ERR_DR[CDNSC] and PCI Express Uncorrectable Status Register[20]) and send interrupt to PIC, if enabled.
Inbound response	CA response for Configuration transaction that originates from ATMU	Log the error (PEX_ERR_DR[PCAC, CDNSC] and PCI Express Uncorrectable Status Register[15]) and send interrupt to PIC, if enabled.
Inbound response	Malformed TLP response	PCI Express controller does not pass the response back to the core. Therefore, a completion timeout error eventually occurs.
Inbound request	Poisoned TLP (EP=1)	1. If it is a posted transaction, the controller drops it. 2. If it is a non-posted transaction, the controller returns a completion with UR status. 3. Release the proper credits
Inbound request	ECRC error	1. If it is a posted transaction, the controller drops it. 2. If it is a non-posted transaction, the controller returns a completion with UR status. 3. Release the proper credits
Inbound request	PCI Express nullified request	The packet is dropped.
Outbound request	Outbound ATMU crossing	Log the error (PEX_ERR_DR[OAC]). The transaction is not sent out on the link.
Outbound request	Illegal message size	Log the error (PEX_ERR_DR[MIS]). The transaction is not sent out on the link.
Outbound request	Illegal I/O size	Log the error (PEX_ERR_DR[IOIS]). The transaction is not sent out on the link.
Outbound request	Illegal I/O address	Log the error (PEX_ERR_DR[IOIA]). The transaction is not sent out on the link.

Table 21-126. Error Conditions (continued)

Transaction Type	Error Type	Action
Outbound request	Illegal configuration size	Log the error (PEX_ERR_DR[CIS]). The transaction is not sent out on the link.
Outbound response	Internal platform response with error (for example, an ECC error on a DDR read or the transaction maps to unknown address space).	Send poisoned TLP (EP=1) completion(s) for data that are known bad. If the poison data happens in the middle of the packet, the rest of the response packet(s) is also poisoned.

21.4.2 Interrupts

Both INTx and message signaled interrupts (MSI) are supported; however there are differences depending on whether the PCI Express controller is configured as an RC or EP device.

21.4.2.1 EP Interrupt Generation

21.4.2.1.1 Hardware INTx Message Generation

Hardware INTx message generation is not supported in EP mode.

21.4.2.1.2 Hardware MSI Generation

In EP mode, the PCI Express controller can be configured to automatically generate MSI transactions to the root complex when an interrupt event occurs. The PCI Express controller uses *irq_out* (an internal version of the IRQ_OUT signal) to trigger the generation of the MSI. To trigger the MSI, interrupt events must be routed to *irq_out* by setting the EP (external pin) bit in the associated Interrupt Destination register in the PIC. Note that the IRQ_OUT signal should not be used for any other function if it is being used to trigger MSI transactions.

The remote root complex is expected to set up the MSI capability structure of all endpoints at system initialization by filling the Message Address and Message Data registers with appropriate values and setting the MSIE bit in the MSI Message Control register.

With the PCI Express controller properly configured, when it detects the leading edge of *irq_out* going active, it generates a PCI Express memory write transaction to the address specified in the MSI Message Address register (and MSI Message Upper Address register) with a data payload as specified in the MSI Message Data register (with leading zeros appended).

21.4.2.1.3 Software INTx Message Generation

Software INTx message generation is not supported in EP mode.

21.4.2.1.4 Software MSI Generation

Host software has to set up the MSI capability registers to enable MSI mode, and have the correct values for the MSI address and data register. Then local software has to 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 has to determine the number of allocated messages in the MSI capability register and

allocates the appropriate data values to use. A write to the ATMU window containing the MSI address with the appropriate data value generates the desired MSI transaction to the remote RC.

21.4.2.2 RC Handling of INTx Message and MSI Interrupts

21.4.2.2.1 INTx Message Handling

MSIs are the preferred interrupt signaling mechanism for PCI Express. However, in RC mode, the PCI Express controller supports the INTx virtual-wire interrupt signaling mechanism (as described in the PCI Express specification). Whenever the controller receives an inbound INTx (INTA, INTB, INTC, or INTD) asserted or negated message, it asserts or negates an equivalent internal INTx signal (*inta*, *intb*, *intc*, or *intd*) to the PIC.

The internal INTx signals from the PCI Express controller are logically combined with the interrupt request (IRQ*n*) input signals so that they share the same interrupt controlled by the associated EIVPR*n* and EIDR*n* registers in the PIC. Refer to Chapter 11, “Programmable Interrupt Controller (PIC),” for more information about handling of PCI Express INTx interrupts and the external interrupt request (IRQ*n*) signals.

If a PCI Express INTx interrupt is being used, then the PIC must be configured so that external interrupts are level-sensitive (EIVPRn[S] = 1).

21.4.2.2.2 MSI Handling

An inbound MSI cycle must hit into the PEXCSRBAR window with the address offset that points to the MSIIR register in the PIC. Note that it is the responsibility of the host software to configure each EP’s MSI capability registers such that an MSI cycle generated from the EP device is routed to the MSIIR register in the PIC and for the appropriate interrupt to be generated to the core.

21.4.3 Initial Credit Advertisement

To prevent overflowing of the receiver’s buffers and for ordering compliance purposes, the transmitter cannot send transactions unless it has enough flow control (FC) credits to send. Each device maintains an FC credit pool. The FC information is conveyed between the two link partners by DLLPs during link training (initial credit advertisement). The transaction layer performs the FC accounting functions. One FC unit is four DWs (16-bytes) of data.

Table 21-127. Initial credit advertisement

Credit Type	Initial Credit Advertisement
PH (Memory Write, Message Write)	4
PD (Memory Write, Message Write)	(256/16)x4=64
NPH (Memory Read, IO Read, Cfg Read, Cfg Write)	4
NPD (IO Write, Cfg Write)	2

Table 21-127. Initial credit advertisement

Credit Type	Initial Credit Advertisement
CPLH (Memory Read Completion, IO R/W Completion, Cfg R/W Completion)	Infinite
CPLD (Memory Read Completion, IO Read Completion, Cfg Read Completion)	Infinite

21.4.4 Power Management

All device power states are supported with the exception of D3cold with Vaux. In addition, all link power states are supported with the exception of L2 states. Only L0s ASPM mode is supported if enabled by configuring the Link Control register's 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 21-128. Power Management State Supported

Component D-State	Permissible Interconnect State	Action
D0	L0, L0s	In full operation.
D1	L0, L0s, L1	All outbound traffics are stalled. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, it is permissible to send a PM_Turn_Off message through the PEX Power Management Command register.
D2	L0, L0s, L1	All outbound traffics are stalled. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, it is permissible to send a PM_Turn_Off message through the PEX Power Management Command register.
D3hot	L0, L0s, L1, L2/L3 Ready	All outbound traffics are stalled. All inbound traffic is thrown away. The only exceptions are PME messages and configuration transactions. If the device is in RC mode, it is permissible to send a PM_Turn_Off message through the PEX Power Management Command register. Note that if a transition of D3hot->D0 occurs, a reset is performed to the controller's configuration space. In addition, link training restarts.
D3cold	L3	Completely off.

21.4.4.1 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 reset or a $\overline{\text{WAKE}}$ signal from the EP device. The PCI Express controller (in EP mode) does not support the generation of beacon; therefore, as an alternative, the device can use one of the GPIO signals as an enable to an external tristate buffer to generate a $\overline{\text{WAKE}}$ signal, as shown in Figure 21-133.

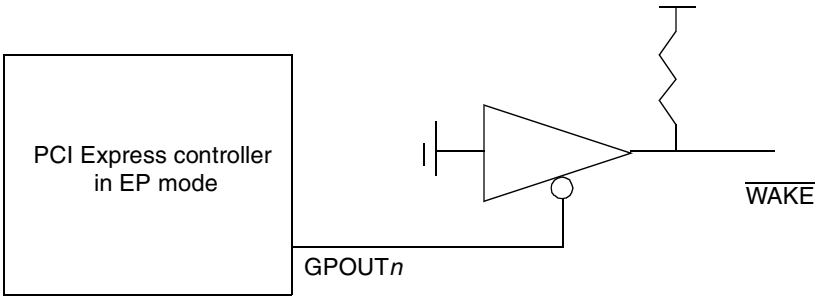


Figure 21-133. $\overline{\text{WAKE}}$ Generation Example

In RC mode, the $\overline{\text{WAKE}}$ signal from the EP device can be connected to one of the external interrupt inputs to service the $\overline{\text{WAKE}}$ request.

21.4.5 Hot Reset

When a hot reset condition occurs, the controller (in both RC and EP mode) initiates a clean-up of all outstanding transactions and returns to an idle state. 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. As an EP device, it is not permitted to generate a hot reset condition; it can only detected a hot reset condition and initiates the clean-up procedure appropriately.

21.4.6 Link Down

Typically, a link down condition occurs after a hot reset event; however, it is possible for the link to go down unexpectedly without a hot reset event. When this occurs, a link down condition is detected (PEX_PME_MSG_DR[LDD]=1). Link down is treated similarly to a hot reset condition.

Subsequently, while the link is down, all new posted outbound transactions are discarded. All new non-posted ATMU transactions are errored out. Non-posted configuration transactions issued using PEX_CONFIG_ADDR/PEX_CONFIG_DATA toward the link returns 0xFFFF_FFFF (all 1s). As soon as the link is up again, the sending of transaction resumes.

Note that in EP mode, a link down condition causes the controller to reset all non-sticky bits in its PCI Express configuration registers as if it had been hot reset.

21.5 Initialization/Application Information

21.5.1 Boot Mode and Inbound Configuration Transactions

In normal boot mode (cfg_cpu_boot = 1), the core is allowed to boot and configure the device. During this time, the PCI Express interface retries all inbound PCI Express configuration transactions. When the core has configured the device to a state where it can accept inbound PCI Express configuration transactions, the boot code should set the CFG_READY bit in the PEX_CFG_READY register after which inbound

PCI Express configuration transactions are accepted. Refer to Section 21.3.10.18, “Configuration Ready Register—0x4B0,” for more information about the CFG_READY bit.

In boot hold-off mode (cfg_cpu_boot = 0), the core is prevented from fetching its first instruction by withholding its internal bus grant. During this time, the PCI Express interface accepts all inbound PCI Express configuration transactions which allows an external host/RC to configure the device. When the external host/RC has configured the device to a state where it can allow the core to fetch code from the boot vector, it sets the PCR[PORT0_EN] bit after which the core is granted the internal bus.

21.5.2 Enabling automatic retraining of link

As part of initialization, software is recommended to clear PEX_FC_UPDATE_TOR[DIS_LR] to allow automatic retraining of link if link partner is down or device has not received DLLP updates within timeout interval specified in PEX_FC_UPDATE_TOR[FC_UPDATE_TIMEOUT].

Chapter 22
General Purpose I/O (GPIO)

This chapter describes theory of operation of the general-purpose I/O (GPIO) module, including a definition of the external signals and functions they serve. Additionally, the configuration, control, and status registers are described. Note that the MPC8610 implements two identical GPIO modules—GPIO1 and GPIO2. Some chapters in this reference manual describe additional specific initialization aspects for each individual block.

22.1 Introduction

This chapter describes the general-purpose I/O module, including signal descriptions, register settings and interrupt capabilities. Figure 22-1 shows the block diagram of the GPIO module.

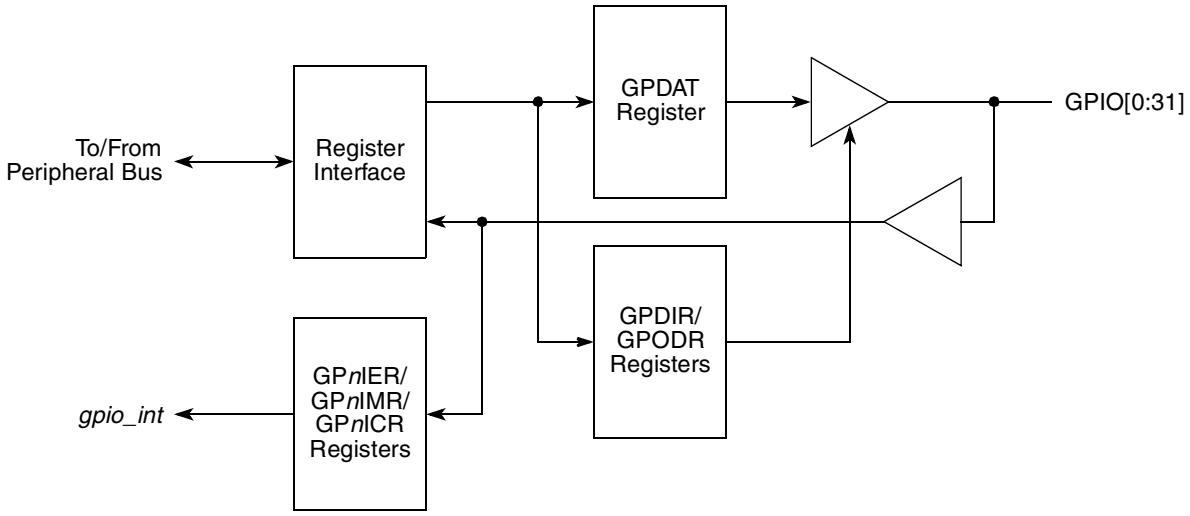


Figure 22-1. GPIO Module Block Diagram

22.1.1 Overview

In general, each GPIO module supports 32 general-purpose I/O ports. Each port can be configured as an input or as an output. However, on the MPC8610, GPIO module 1 supports 16 ports that can be input or output and 16 ports that must be output-only; GPIO module 2 supports 26 ports that can be input or output and 1 port that must be input-only. If a port is configured as an input, it can optionally generate an interrupt upon 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.

22.1.2 Features

The GPIO unit implements the following features:

- GPIO1 has 32 ports (16 input/output and16 output-only)
- GPIO2 has 27 ports (26 input/output and 1 input-only)
- All ports are multiplexed with other functional signals
- All I/O signals are configured as inputs when the device comes out of reset and also when $\overline{\text{HRESET}}$ signal is asserted
- Open-drain capability on all ports
- All ports can optionally generate an interrupt upon changing their state

22.2 External Signal Descriptions

Table 22-1 provides detailed descriptions of the external GPIO signals. Note that depending on the signal multiplexing, some signals may not be available.

Table 22-1. GPIO External Signals—Detailed Signal Descriptions

Signal	I/O	Description	
GPIO1[0:15], GPIO2[5:10], GPIO2[12:31]	I/O	General Purpose I/O. Each pin can be individually set to act as input or output, according to application needs.	
		State Meaning	Asserted/Negated—Defined per application.
		Timing	Assertion/Negation —Inputs can be asserted completely asynchronously. Outputs are asynchronous to any externally visible clock.
GPIO1[16:31]	O	General Purpose I/O. These pins are output-only.	
		State Meaning	Asserted/Negated—Defined per application.
		Timing	Assertion/Negation —Outputs are asynchronous to any externally visible clock.
GPIO2[11]	I	General Purpose I/O. This pin is input-only.	
		State Meaning	Asserted/Negated—Defined per application.
		Timing	Assertion/Negation —Input can be asserted asynchronously.

22.3 Memory Map/Register Definition

The programmable register map for each GPIO module occupies 24 bytes of memory-mapped space. The full register address is comprised of the CCSR window base address, specified in CCSRBAR, plus the functional block base address, plus the specific register's offset within that block. The block base address for GPIO is 0x0_F000. The registers for GPIO1 are at offsets 0x000–0x017 and the registers for GPIO2 are at offsets 0x100–0x117.Table 22-2 shows the memory map for the GPIO modules.

All GPIO registers are 32 bits wide located on 32-bit address boundaries. Note that reading undefined portions of the memory map returns all zeros and writing has no effect.

Table 22-2. GPIO Register Address Map

GPIO Registers—Block Base Address 0x0_F000				
Offset	**Register**	**Access**	**Reset Value**	**Section/Page**
GPIO1 Registers				
0x000	GPIO direction register (GPDIR)	R/W	All zeros	22.3.1/22-3
0x004	GPIO open drain register (GPODR)	R/W	All zeros	22.3.2/22-4
0x008	GPIO data register (GPDAT)	R/W	All zeros	22.3.3/22-4
0x00C	GPIO interrupt event register (GPIER)	w1c	Undefined	22.3.4/22-5
0x010	GPIO interrupt mask register (GPIMR)	R/W	All zeros	22.3.5/22-5
0x014	GPIO interrupt control register (GPICR)	R/W	All zeros	22.3.6/22-6
GPIO2 Registers				
0x100–0x114	Same as GPIO1 but offset by 0x100			

22.3.1 GPIO Direction Register (GPDIR)

The GPIO direction register (GPDIR), shown in Figure 22-2, defines the direction of the individual ports.

Offset: 0x000 (GPIO1)
0x100 (GPIO2)

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DR0	DR1	DR2	DR3	DR4	DR5	DR6	DR7	DR8	DR9	DR10	DR11	DR12	DR13	DR14	DR15
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DR16	DR17	DR18	DR19	DR20	DR21	DR22	DR23	DR24	DR25	DR26	DR27	DR28	DR29	DR30	DR31
Reset	All zeros															

Figure 22-2. GPIO Direction Register (GPDIR)

Table 22-3 defines the bit fields of GPDIR.

Table 22-3. GPDIR Field Descriptions

Bits	Name	Description
0–31	DR*n*	Direction. Indicates whether a pin is used as an input or an output. **Note:** GPIO1[16:31] are always outputs, GPIO2[11] is always an input., and GPIO2[0:4] do not exist. 0 The corresponding pin is an input. 1 The corresponding pin is an output.

22.3.2 GPIO Open Drain Register (GPODR)

The GPIO Open Drain Register (GPODR), shown in Figure 22-3, defines the way individual ports drive their output.

Offset: 0x004 (GPIO1)
0x104 (GPIO2)

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	OD0	OD1	OD2	OD3	OD4	OD5	OD6	OD7	OD8	OD9	OD10	OD11	OD12	OD13	OD14	OD15
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OD16	OD17	OD18	OD19	OD20	OD21	OD22	OD23	OD24	OD25	OD26	OD27	OD28	OD29	OD30	OD31
Reset	All zeros															

Figure 22-3. GPIO Open Drain Register (GPODR)

Table 22-4 defines the bit fields of GPODR.

Table 22-4. GPODR Field Descriptions

Bits	Name	Description
0–31	OD*n*	Open drain configuration. Indicates whether a signal is actively driven as an output or is an open-drain driver. This register has no effect on signals programmed as inputs in GPDIR. 0 The corresponding signal is actively driven as an output. 1 The corresponding signal is an open-drain driver. As an output, the signal is driven active-low, otherwise it is not driven (high impedance).

22.3.3 GPIO Data Register (GPDAT)

The GPIO data register (GPDAT), shown in Figure 22-4 carries the data in/out for the individual ports.

Offset: 0x008 (GPIO1)
0x108 (GPIO2)

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
Reset	All zeros															

Figure 22-4. GPIO Data Register (GPDAT)

Table 22-5 defines the bit fields of GPDAT.

Table 22-5. GPDAT Field Descriptions

Bits	Name	Description
0–31	D*n*	Data. Writes to this register latches the data which is presented on the external pins provided the corresponding GPDIR bit is configured as an output. Read operations always return the data at the pin.

22.3.4 GPIO Interrupt Event Register (GPIER)

The GPIO interrupt event register (GPIER), shown in Figure 22-5 carries information of the events that caused an interrupt. Each bit in GPIER, corresponds to an interrupt source. GPIER bits are cleared by writing ones. However, writing zero has no effect.

Offset: 0x00C (GPIO1)
0x10C (GPIO2)

Access: w1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EV0	EV1	EV2	EV3	EV4	EV5	EV6	EV7	EV8	EV9	EV10	EV11	EV12	EV13	EV14	EV15
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	Undefined (the user should write 1s to clear before using)															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EV16	EV17	EV18	EV19	EV20	EV21	EV22	EV23	EV24	EV25	EV26	EV27	EV28	EV29	EV30	EV31
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	Undefined (the user should write 1s to clear before using)															

Figure 22-5. GPIO Interrupt Event Register (GPIER)

Table 22-6 defines the bit fields of GPIER.

Table 22-6. GPIER Field Descriptions

Bits	Name	Description
0–31	EV*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 An interrupt event occurred on the corresponding GPIO signal.

22.3.5 GPIO Interrupt Mask Register (GPIMR)

The GPIO interrupt mask register (GPIMR), shown in Figure 22-6 defines the interrupt masking for the individual ports. When a masked interrupt request occurs, the corresponding GPIER bit is set, regardless of the GPIMR state. When one or more non-masked interrupt events occur, the GPIO module issues an interrupt to the interrupt controller.

Offset: 0x010 (GPIO1)
0x110 (GPIO2)

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IM0	IM1	IM2	IM3	IM4	IM5	IM6	IM7	IM8	IM9	IM10	IM11	IM12	IM13	IM14	IM15
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IM16	IM17	IM18	IM19	IM20	IM21	IM22	IM23	IM24	IM25	IM26	IM27	IM28	IM29	IM30	IM31
Reset	All zeros															

Figure 22-6. GPIO Interrupt Mask Register (GPIMR)

Table 22-7 defines the bit fields of GPIMR.

Table 22-7. GPIMR Field Descriptions

Bits	Name	Description
0–31	IM*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 An interrupt event occurred on the corresponding GPIO signal.

22.3.6 GPIO Interrupt Control Register (GPICR)

The GPIO interrupt control register (GPICR), shown in Figure 22-7 determines whether the corresponding port line asserts an interrupt request upon either a high-to-low change or any change on the state of the signal.

Offset: 0x014 (GPIO1)
0x114 (GPIO2)

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ED0	ED1	ED2	ED3	ED4	ED5	ED6	ED7	ED8	ED9	ED10	ED11	ED12	ED13	ED14	ED15
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ED16	ED17	ED18	ED19	ED20	ED21	ED22	ED23	ED24	ED25	ED26	ED27	ED28	ED29	ED30	ED31
Reset	All zeros															

Figure 22-7. GPIO Interrupt Control Register (GPIMR)

Table 22-8 defines the bit fields of GPICR.

Table 22-8. GPICR Field Descriptions

Bits	Name	Description
0–31	ED*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.

Part IV
Global Functions and Debug

Part IV defines other global blocks of the MPC8610. The following chapters are included:

- Chapter 23, “Global Utilities,” defines the global utilities of the MPC8610. These include power management, I/O device enabling, power-on-reset (POR) configuration monitoring, general-purpose I/O signal use, and multiplexing for the interrupt and local bus chip select signals.
- Chapter 24, “Device Performance Monitor,” describes the performance monitor of the MPC8610.
- Chapter 25, “Debug Features and Watchpoint Facilities,” describes the debug features and watchpoint monitor of the MPC8610.

Chapter 23
Global Utilities

This chapter describes the global utilities of the MPC8610. It provides signal descriptions, register descriptions, and a functional description of these utilities.

23.1 Overview

The global utilities block controls power management, I/O device enabling, power-on-reset (POR) configuration monitoring, alternate function selection for multiplexed signals, and clock control.

23.2 Global Utilities Features

This section provides an overview of global utilities features.

23.2.1 Power Management and Block Disables

The following features affect the device's overall power consumption:

- Software-controlled power management (core nap and core sleep, controlled by bits in the core's HID0 register; platform sleep, controlled by POWMGTCSR[SLEEP]).
- Static power management (I/O block disables)

23.2.2 Accessing Current POR Configuration Settings

The POR configuration values of all device parameters sampled from pins at reset are available through memory-mapped registers in the global utilities block.

23.2.3 Signal Multiplexing Controls

Many signals have alternate functions that are selected by memory-mapped registers in the global utilities block.

23.2.4 Clock Control

The global utilities block also selects the internal clock signal driven on CLK_OUT.

23.3 External Signal Descriptions

The following sections provide information about signals that serve as global utilities.

23.3.1 Signals Overview

Table 23-1 summarizes the external signals used by the global utilities block

Table 23-1. External Signal Summary

Signal Name	I/O	Description	Reference (Section/Page)
ASLEEP	O	Signals that the device has reached a sleep state.	—
CLK_OUT	O	Clock out. Selected by CLKOCR values.	23.4.1.25/23-29
CKSTP_ IN	I	Checkstop input.	Table 23-2 on page 23-2
CKSTP_OUT	O	Checkstop output.	Table 23-2 on page 23-2
SMI	I	System Management Interrupt input for e600 core	Table 23-2 on page 23-2

23.3.2 Detailed Signal Descriptions

Table 23-2 describes signals in the global utilities block in detail.

Table 23-2. Detailed Signal Descriptions

Signal	I/O	Description	
ASLEEP	O	Asleep. After negation of $\overline{\text{HRESET}}$, ASLEEP is asserted until the device completes its power-on reset sequence and reaches its ready state.	
		State Meaning	Asserted—Indicates that the device is either still in its power-on reset sequence or it has reached a sleep state after a power-down command is issued by software. Negated—The device is not in sleep mode. (It has either awakened from a power-down state, or has completed the POR sequence.)
		Timing	Assertion—May occur at any time; may be asserted asynchronously to the input clocks. Negation—Negates synchronously with SYSCLK when leaving power-on sequence; otherwise negation is asynchronous.
$\overline{\text{CKSTP\_IN}}$	I	Checkstop in	
		State Meaning	Asserted—Indicates that the e600 core must enter a hard stop condition. All e600 clocks are turned off. $\overline{\text{CKSTP\_OUT}}$ is asserted. The rest of device logic, including memory controllers, internal memories and registers, and I/O interfaces, remains functional. Negated—Indicates that normal operation should proceed.
		Timing	Assertion—May occur at any time; may be asserted asynchronously to the input clocks. Negation—Must remain asserted until the device is reset with assertion of $\overline{\text{HRESET}}$.

Table 23-2. Detailed Signal Descriptions (continued)

Signal	I/O	Description	
CKSTP_OUT	O	Checkstop out	
		State Meaning	Asserted—Indicates that the e600 core of the device is in a checkstop state. The rest of the device logic remains functional. Negated—Indicates normal operation. After $\overline{\text{CKSTP\_OUT}}$ has been asserted, it is negated after the next negation (low-to-high transition) of $\overline{\text{HRESET}}$.
		Timing	Assertion—May occur at any time; may be asserted asynchronously to the input clocks. Negation—Must remain asserted until the device has been reset with a hard reset.
CLK_OUT	O	Clock out. Reflects clock signal selected by CLKOCR (see Section 23.4.1.25, "CLK_OUT Control Register (CLKOCR)").	
		State Meaning	Asserted—If CLKOCR[ENB] = 1, clock signal selected by CLKOCR[CLK_SEL] is driven. High impedance—If CLKOCR[ENB] = 0.
		Timing	Assertion/Negation—Depends on the value of CLKOCR[CLK_SEL].
$\overline{\text{SMI}}$	I	System Management Interrupt for e600 core. See interrupt section of the *e600 Core Reference Manual* for additional information	
		State Meaning	Asserted—Indicates that the e600 core should take a System Management Interrupt exception if enabled in the MSR. Negated—Indicates that there is no pending System Management exception.
		Timing	Assertion—May occur at any time; may be asserted asynchronously to the input clocks. $\overline{\text{SMI}}$ is level-sensitive. Negation—Must not occur until after the interrupt is taken.

23.4 Memory Map/Register Definition

The global utilities registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR), plus the block base address, plus the offset of the specific register to be accessed. For the global utilities registers, the block base address is 0xE_0000.

Table 23-3 summarizes the global utilities registers and their addresses.

Table 23-3. Global Utilities Register Map

Offset	Register	Access	Reset [1]	Section/Page
Global Utilities Registers—Block Base Address 0xE_0000				
Power-On Reset Configuration Values				
0x000	PORPLLSR—POR PLL ratio status register	R	0x00*nn*_*n*0*nn*	23.4.1.1/23-5
0x004	PORBMSR—POR boot mode status register	R	0x*nnnn*_0000	23.4.1.2/23-6
0x008	PORIMPCR—POR I/O impedance control register	Mixed	0x000*n*_007F	23.4.1.3/23-8
0x00C	PORDEVSR—POR I/O device status register	R	see ref***.	23.4.1.4/23-9
0x010	PORDBGMSR—POR debug mode status register	R	see ref.	23.4.1.5/23-10
0x018	PORGEN—POR general usage register	R	0x*nn*0*n*_0000	23.4.1.6/23-10

Table 23-3. Global Utilities Register Map (continued)

Offset	Register	Access	Reset [1]	Section/Page
0x020	PORCIR—POR configuration information register	R	see ref.	23.4.1.7/23-11
Signal Multiplexing and GPIO Controls				
0x030	GPIOCR—GPIO control register	R/W	All zeros	23.4.1.8/23-12
0x060	PMUXCR—Alternate function signal multiplex control	R/W	All zeros	23.4.1.9/23-14
Device Disables				
0x070	DEVDISR—Device disable control	R/W	All zeros	23.4.1.10/23-16
0x074	DEVDISR2—Device disable control 2	R/W	All zeros	23.4.1.10/23-16
Power Management Registers				
0x080	POWMGTCSR—Power management status and control register	Mixed	All zeros	23.4.1.12/23-19
Interrupt and Reset Status and Control Registers				
0x090	MCPSUMR—Machine check summary register	w1c	All zeros	23.4.1.13/23-20
0x094	RSTRSCR—Reset request status and control register	R	All zeros	23.4.1.14/23-21
Version Registers				
0x0A0	PVR—Processor version register	R	0x8004_*nnnn*	23.4.1.15/23-21
0x0A4	SVR—System version register	R	0x80A0_00*nn*	23.4.1.16/23-22
Reset Registers				
0x0B0	RSTCR—Reset control register	R/W	All zeros	23.4.1.17/23-22
0x0C0	eLBCVSELCR—eLBC voltage select control register	R/W	All zeros	23.4.1.18/23-23
Clock Control				
0x800	CKDVDR—Clock divide register	R/W	All zeros	23.4.1.19/23-23
Configuration and Status				
0x900	IRCR—Infrared control register	R/W	All zeros	23.4.1.20/23-25
0x904	MCMCR—MCM control register	R/W	All zeros	23.4.1.21/23-25
0x908	DMACR—DMA control register	R/W	All zeros	23.4.1.22/23-26
0x914	eLBCCR—eLBC control register	R/W	All zeros	23.4.1.23/23-27
DDR Control				
0xB28	DDRCLKDR—DDR clock disable register	R/W	All zeros	23.4.1.24/23-28
Debug, Clock, and SerDes Control				
0xE00	CLKOCR—CLK_OUT control register	R/W	All zeros	23.4.1.25/23-29
0xF04	SRDS1DCR0—SerDes1 (×4) debug control register 0	R/W	0x1105_4418	23.4.1.26/23-30

Table 23-3. Global Utilities Register Map (continued)

Offset	Register	Access	Reset [1]	Section/Page
0xF08	SRDS1DCR1—SerDes2 (×4) debug control register 1	R/W	0x0000_000*n*	23.4.1.27/23-31
0xF40	SRDS2DCR0—SerDes2 (×8) debug control register 0	R/W	0x1105_4418	23.4.1.28/23-31
0xF44	SRDS2DCR1—SerDes2 (×8) debug control register 1	R/W	0x0000_000*n*	23.4.1.29/23-32

[1] Bits indicated with *n* are set from configuration signals.

23.4.1 Register Descriptions

This section describes the global utilities registers in detail.

23.4.1.1 POR PLL Status Register (PORPLLSR)

PORPLLSR, shown in Figure 23-1, contains the settings for the PLL ratios as set by the cfg_sys_pll[0:4], cfg_core_pll[0:3], and POR configuration pins.

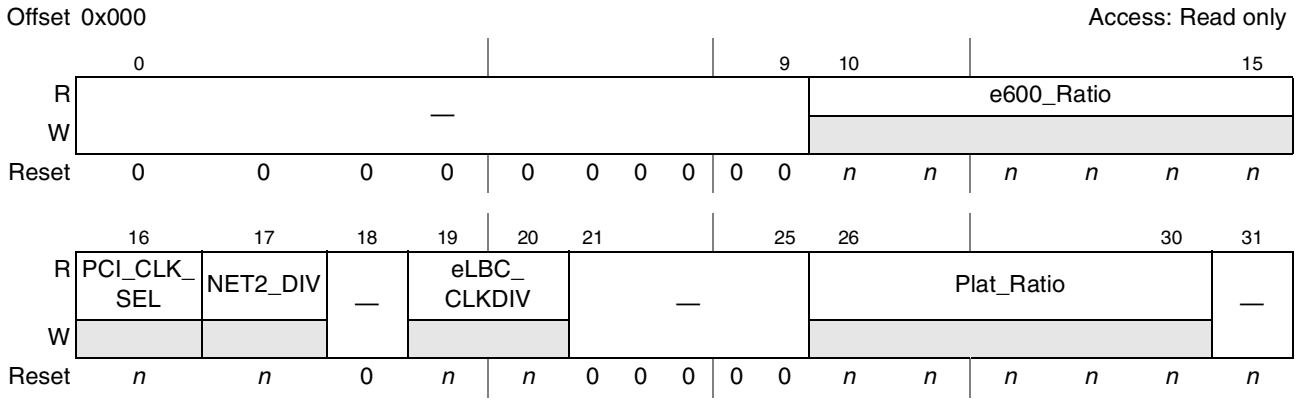


Figure 23-1. POR PLL Status Register (PORPLLSR)

Table 23-4 describes the bit settings of PORPLLSR.

Table 23-4. PORPLLSR Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10–15	e600_Ratio	Clock ratio between the e600 core and the platform (MPX bus) clock. Values not shown are reserved. 010000 2:1 011001 5:2 (2.5:1) 100000 3:1 111001 7:2 (3.5:1) Any other combinations besides the ones listed are not supported for this device

Table 23-4. PORPLLSR Field Descriptions (continued)

Bits	Name	Description
16	PCI_CLK_SEL	PCI Clock Select. (See Section 4.4.3.14, "PCI Clock Select.") 0 Asynchronous mode. PCI interface is clocked by PCI_CLK input. 1 Synchronous mode. PCI interface is clocked by SYSCLK. **Note:** 2:1 platform clock ratio not supported in synchronous mode.
17	NET2_DIV	OCN2 clock ratio divider. NET2_DIV establishes the clock ratio between the on-chip network #2 (OCN2) and the platform clock used by the device. (See Section 4.4.3.5, "OCN2 Ratio.") 0 1:1 1 1:2. A 1:2 ratio is required whenever the platform frequency is greater than 400 MHz.
18	—	Reserved
19–20	eLBC_CLKDIV	eLBC clock divider. Specifies the platform clock divisor used by the eLBC. 00 eLBC clock is set to platform clock/4 LCRR[CLKDIV] = 00010 01 eLBC clock is set to platform clock/8 LCRR[CLKDIV] = 00100 10 Reserved 11 eLBC clock is set to platform clock/16 LCRR[CLKDIV] = 01000 **Note:** For GPCM booting, the maximum division of /16 is recommended. **Note:** For NAND-Flash booting through the FCM, the ratio is typically selected so that 66 MHz operation is not exceeded. That is: /4 (00) for platform frequencies up to 333 MHz /8 (01) for platform frequencies up to 533 MHz
16–25	—	Reserved
26–30	Plat_Ratio	Clock ratio between the platform (MPX) clock and SYSCLK. Values not shown are reserved. (See Section 4.4.3.3, "Platform PLL Ratio.") 00011 3:1 00100 4:1 00101 5:1 00110 6:1 00111 7:1 01000 8:1 01001 9:1 01010 10:1 01100 12:1 01110 14:1 01111 15:1 10000 16:1
31	—	Reserved

23.4.1.2 POR Boot Mode Status Register (PORBMSR)

The PORBMSR, shown in Figure 23-2, reports setting of the POR configuration pins that control the boot mode settings.

Offset 0x004 Access: Read only

	0	1	2	3	4			7	8	9	10	11	12	13	14	15															31
R	BCFG		—		ROM_LOC				—		BSCFG		—	HA			—														ABV
W																															
Reset	*n*	0	0	0	*n*	*n*	*n*	*n*	0	0	*n*	*n*	0	*n*	*n*	*n*	0	0	0	0	0	0	0	0	0	0	0	0	0	0	*n*

Figure 23-2. POR Boot Mode Status Register (PORBMSR)

Figure 23-5 describes the bit settings of the PORBMSR.

Table 23-5. PORBMSR Field Description

Bits	Name	Description
0	BCFG	CPU boot configuration. (See Section 4.4.3.6, "CPU Boot Configuration.") 0 The e600 core is prevented from booting until configured by an external master. In this case, the external master must also set PCR [PORT0_EN] in the MCM block when the external master is ready for e600 core to boot. 1 e600 core is allowed to boot without waiting for configuration by an external master.
1–3	—	Reserved
4–7	ROM_LOC	Location of boot ROM. This field indicates the location of the boot ROM, as determined by the values on cfg_rom_loc[0:3] at the negation of $\overline{\text{HRESET}}$. (See Section 4.4.3.8, "Boot ROM Location.") 0000–0011Reserved 0100 eLBC, FCM, small page, 8-bit NAND Flash ROM 0101 eLBC, FCM, small page, 16-bit NAND Flash ROM 0110 eLBC, FCM, large page, 8-bit NAND Flash ROM 0111 eLBC, FCM, large page, 16-bit NAND Flash ROM 1000 PCI 1001 DDR Controller 1010 Reserved 1011 PCI Express 2 (×8) 1100 PCI Express 1 (×4) 1101 eLBC, GPCM, 8-bit Flash ROM 1110 eLBC, GPCM, 16-bit Flash ROM 1111 eLBC, GPCM, 32-bit Flash ROM (default)
8–9	—	Reserved
10–11	BSCFG	Boot sequencer configuration. (See Section 4.4.3.7, "Boot Sequencer Configuration.") 00 Reserved 01 Boot sequencer enabled with normal I^2C addressing 10 Boot sequencer enabled with extended I^2C addressing 11 Boot sequencer disabled
12	—	Reserved
13-15	HA	Host/agent mode configuration. (See Section 4.4.3.11, "Host/Agent Configuration.") ValuePCIPCI Express 1 (×4)PCI Express 2 (×8) 000HostEPEP 001HostRCEP 010HostEPRC 011Reserved 100AgentRCEP 101AgentEPRC 110AgentRCRC 111HostRCRC
16-30	—	Reserved
31	ABV	Alternate Boot Vector. See Section 4.4.3.10, "Alternate Boot Vector," for more information. 0 Alternate Boot Vector mode is enabled. Outbound Window 1 in PCI Express 1 (×4) is automatically enabled to translate addresses matching 0x0_FFF0_*nnnn* to 0x0_FFFF_*nnnn*. Used with PORBMSR[ROM_LOC]=PCI Express 1 to steer the e600 boot vector. 1 Alternate Boot Vector mode is disabled. CPU boots from the default boot vector (0x0_FFF0_*nnnn*)

23.4.1.3 POR I/O Impedance Control Register (PORIMPCR)

PORIMPCR, shown in Figure 23-3, contains the I/O driver impedance select for the PCI and local bus interfaces.

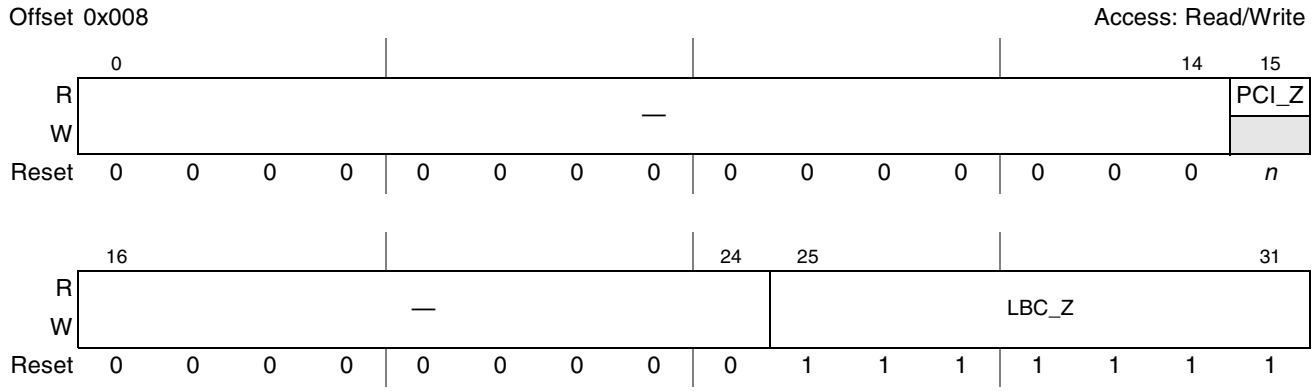


Figure 23-3. POR I/O Impedance Control Register (PORIMPCR)

The I/O impedance of the PCI bus and the local bus signals (including the local bus clock) is controlled through this register. The *MPC8610 Integrated Processor Hardware Specification* provides exact I/O impedances.

Table 23-6 describes PORIMPCR fields.

Table 23-6. PORIMPCR Field Descriptions

Bits	Name	Description
0–14	—	Reserved
15	PCI_Z	PCI bus impedance. (See Section 4.4.3.16, “PCI I/O Impedance.”) 0 Low impedance drivers 1 High-impedance drivers
16–24	—	Reserved
25-31	LBC_Z	I/O impedance for Group 1 enhanced local bus signals 111111 High impedance All other values low impedance Note that Group 1 enhanced local bus signals consist of LALE, LAD[0:31], LDP[0:3], LA[10:31], LCKE, LCS[1:2]; Other eLBC signals use a fixed high impedance.

23.4.1.4 POR Device Status Register (PORDEVSR)

Shown in Figure 23-4, PORDEVSR reports other POR settings for I/O devices.

Offset 0x00C Access: Read only

	0							7	8	9	10			12	13	14	15
R	—								LA_SEL	—	IO_SEL			—		PCI_ARB	—
W																	
Reset	0	0	0	0	0	0	0	0	*n*	0	*n*	*n*	*n*	0		*n*	0

	16	17	18					24	25	26	27					31
R	PCI_SPD	WDT_EN	—						RTYPE		—					
W																
Reset	*n*	*n*	0	0	0	0	0	0	*n*	*n*	0	0	0	0	0	0

Figure 23-4. POR Device Status Register (PORDEVSR)

Table 23-7 describes the bit settings of PORDEVSR.

Table 23-7. PORDEVSR Field Descriptions

Bits	Name	Description
0–7	—	Reserved
8	LA_SEL	Latched address selection mode. See Section 4.4.3.19, “LA Function Configuration.” 0 LA[10:21] are used for eLBC 1 LA[10:21] are used for SSI
9	—	Reserved
10–12	IO_SEL	I/O port selection mode (See Section 4.4.3.12, “SerDes (I/O) Port Selection.”) 000 SerDes port 2 (PCI Express 2) active 001 SerDes port 1 (PCI Express 1) active 010 Reserved 011 Reserved 100 Both SerDes ports active 101 Reserved 110 Reserved 111 Both SerDes ports disabled
13	—	Reserved
14	PCI_ARB	PCI arbiter enable (See Section 4.4.3.17, “PCI Arbiter.”) 0 On-chip PCI arbiter is disabled 1 On-chip PCI arbiter is enabled
15	—	Reserved
16	PCI_SPD	PCI bus speed (See Section 4.4.3.15, “PCI Speed.”) 0 PCI bus is operating below 33 MHz 1 PCI bus is operating at or above 33 MHz
17	WDT_EN	Watchdog timer enable. (See Section 4.4.3.18, “Watchdog Timer Enable.”) 0 Watchdog timer is disabled out of reset 1 Watchdog timer is enabled out of reset
18–24	—	Reserved

Table 23-7. PORDEVSR Field Descriptions (continued)

Bits	Name	Description
25–26	RTYPE	DRAM Type (See Section 4.4.3.13, “DDR SDRAM Type.”) 00 Reserved 01 DDR1, 2.5V, CKE low at reset 10 Reserved 11 DDR2, 1.8V, CKE low at reset
27–31	—	Reserved

23.4.1.5 POR Debug Mode Status Register (PORDBGMSR)

PORDBGMSR, shown in Figure 23-5, holds debug mode settings from the POR configuration pins as described in Section 4.4.3.21, “Memory Debug Configuration.”

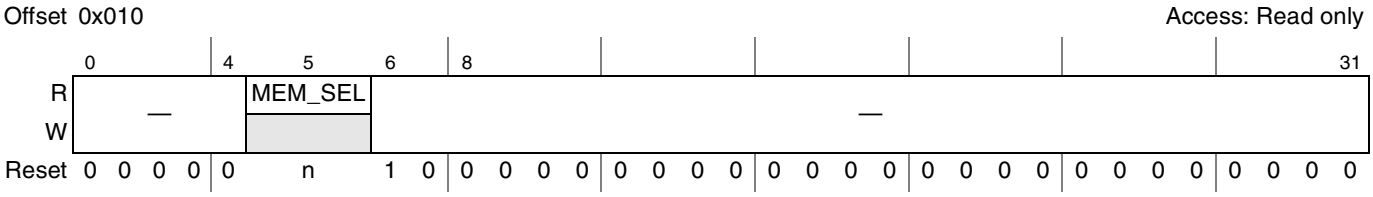


Figure 23-5. POR Debug Mode Status Register (PORDBGMSR)

Table 23-8 describes the bit settings of PORDBGMSR.

Table 23-8. PORDBGMSR Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5	MEM_SEL	Memory select. Indicates which controller is driving MSRCID[0:4] and MDVAL. 0 eLBC is driving debug information 1 DDR controller is driving debug information
6–31	—	Reserved

23.4.1.6 POR General Register (PORGEN)

PORGEN, shown in Figure 23-6, reports setting of the cfg_elbc_ecc POR configuration pin that controls the boot mode setting for ECC on the eLBC interface.

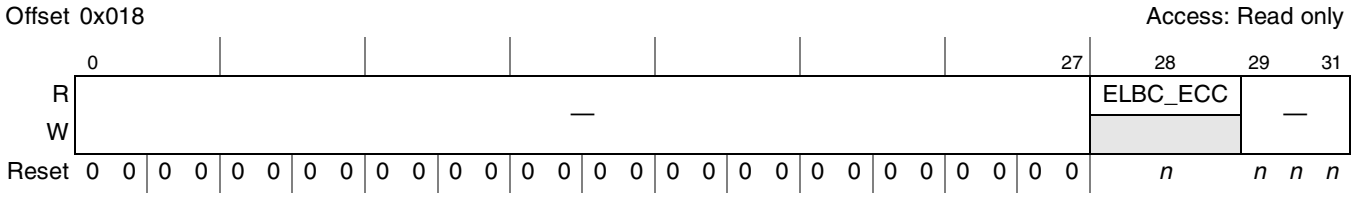


Figure 23-6. POR General Register (PORGEN)

Table 23-9 describes the bit settings of PORGEN.

Table 23-9. PORGEN Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28	ELBC_ECC	eLBC ECC configuration. (See Section 4.4.3.9, “eLBC ECC Enable.”) 0 eLBC ECC checking is disabled (BR0[DECC] = 00) 1 eLBC ECC checking enabled (BR0[DECC] = 01)
29–31	—	Reserved

23.4.1.7 POR Configuration Information Register (PORCIR)

Shown in Figure 23-7, PORCIR stores the value sampled from the local bus address/data signals, LAD[0:31], during POR, as described in Section 4.4.3.20, “General-Purpose POR Configuration.” Software can use this value to inform the operating system about initial system configuration. Typical interpretations include circuit board type, board ID number, or a list of available peripherals.

NOTE

This read-only register is not for use as general-purpose input/output (GPIO). It is intended to allow system designers to pass (for example) a device-specific version value. This happens one time only, at reset.

True general-purpose I/O is available. See the registers of Section 22.5, “Memory Map/Register Definition,” for more information.

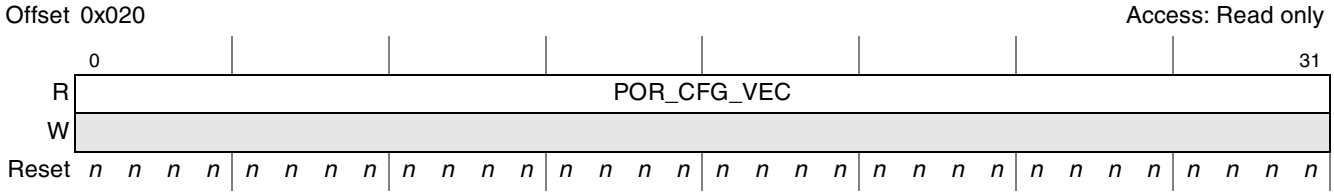


Figure 23-7. POR Configuration Information Register (PORCIR)

Table 23-10 describes the bit settings of PORCIR.

Table 23-10. PORCIR Field Descriptions

Bits	Name	Description
0–31	POR_CFG_VEC	POR configuration vector sampled from local bus address/data signals LAD[0:31]at the negation of $\overline{\text{HRESET}}$. Note that if nothing is driven on these signals during reset, the value of this register is indeterminate.

23.4.1.8 General-Purpose I/O Control Register (GPIOCR)

Shown in Figure 23-8, GPIOCR contains the enable bits for each group of pins that may be used for general-purpose I/O. These bits have meaning only if the signals are not being used for their primary function.

Offset 0x030 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PCI_RG1	PCI_RG2	PCI_RG3	PCI_RG4	GTM1_1	—	GTM1_3	—	GTM2_1	—	DIU8	—	DIU16	—	IR1	—
Reset	All zeros															

	16	17	18	19	20	21	22	23	24	25	26	27	31
R W	UART0		UART2		UART1		UART3		I2C1	I2C2_SCL	I2C2_SDA	—	
Reset	All zeros												

Figure 23-8. General-Purpose I/O Control Register (GPIOCR)

Table 23-11 describes the bit settings of GPIOCR.

Table 23-11. GPIOCR Field Descriptions

Bits	Name	Description
0	PCI_RG1	0 GPIO1[0:1] 1 $\overline{\text{PCI\_REQ1}}$, $\overline{\text{PCI\_GNT1}}$ **Note:** PCI input pins are tied as deasserted when not selected.
1	PCI_RG2	0 GPIO1[2:3] 1 $\overline{\text{PCI\_REQ2}}$, $\overline{\text{PCI\_GNT2}}$ **Note:** PCI input pins are tied as deasserted when not selected.
2	PCI_RG3	0 GPIO1[4:5] 1 $\overline{\text{PCI\_REQ3}}$, $\overline{\text{PCI\_GNT3}}$ **Note:** PCI input pins are tied as deasserted when not selected.
3	PCI_RG4	0 GPIO1[6:7] 1 $\overline{\text{PCI\_REQ4}}$, $\overline{\text{PCI\_GNT4}}$ **Note:** PCI input pins are tied as deasserted when not selected.
4	GTM1_1	0 GPIO2[15:17] 1 GTM1_TIN1, GTM1_$\overline{\text{TGATE}}$1, GTM1_$\overline{\text{TOUT}}$1 **Note:** GTM input pins are tied as deasserted when not selected.
5	—	Reserved
6	GTM1_3	0 GPIO2[21:23] 1 GTM1_TIN3, GTM1_$\overline{\text{TGATE}}$3, GTM1_$\overline{\text{TOUT}}$3 **Note:** GTM input pins are tied as deasserted when not selected.
7	—	Reserved
8	GTM2_1	0 GPIO2[18:20] 1 GTM2_TIN1, GTM2_$\overline{\text{TGATE}}$1, GTM2_$\overline{\text{TOUT}}$1 **Note:** GTM input pins are tied as deasserted when not selected.

Table 23-11. GPIOCR Field Descriptions (continued)

Bits	Name	Description
9	—	Reserved
10	DIU8	0 8 muxed pins used for GPIO1[15:8] 1 8 muxed pins used for DIU_LD[23:16] **Note:** If DIU8 and DIU16 bits are both set for GPIO then the following pins are tri-stated: DIU_VSYNC, DIU_HSYNC, DIU_DE.
11	—	Reserved
12	DIU16	0 16 muxed pins used for GPIO1[31:16] 1 16 muxed pins used for DIU_LD[15:0] **Note:** If DIU8 and DIU16 bits are both set for GPIO then the following pins are tri-stated: DIU_VSYNC, DIU_HSYNC, DIU_DE.
13	—	Reserved
14	IR1	0 GPIO2[13:14] 1 IR1_TXD, IR1_RXD **Note:** IR input pins are tied as deasserted when not selected.
15	—	Reserved
16–17	UART0[2]	00 GPIO2[5] 01 UART0's SOUT/SIN 10 Reserved 11 SPI[1] **Note:** If 00 is selected, the UART_SOUT0 pin is tristated. **Note:** UART/SPI input pins are tied as deasserted when not selected.
18–19	UART2	00 GPIO2[6] 01 UART0's $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ 10 Reserved 11 UART2's SOUT/SIN **Note:** If 00 is selected, the $\overline{\text{UART\_RTS0}}$ pin is tristated. **Note:** UART input pins are tied as deasserted when not selected.
20–21	UART1	00 GPIO2[7] 01 UART1's SOUT/SIN 10 Reserved 11 IR2 **Note:** If 00 is selected, the UART_SOUT1 pin is tristated. **Note:** UART/IR input pins are tied as deasserted when not selected.
22–23	UART3	00 GPIO2[8] 01 UART1's $\overline{\text{RTS}}$/$\overline{\text{CTS}}$ 10 Reserved 11 UART3's SOUT/SIN **Note:** If 00 is selected, the $\overline{\text{UART\_RTS1}}$ pin is tristated. **Note:** UART input pins are tied as deasserted when not selected.
24	I2C1	0 I2C1 1 GPIO2[9:10] **Note:** I2C/SPI input pins are tied as deasserted when not selected.
25	I2C2_SCL[1]	0 I2C2 1 GPIO2[11] **Note:** I2C/SPI input pins are tied as deasserted when not selected.

Table 23-11. GPIOCR Field Descriptions (continued)

Bits	Name	Description
26	I2C2_SDA[2]	0 I2C2 / SPI's $\overline{\text{SPISEL}}$ 1 GPIO2[12] **Note:** When GPIOCR[UART0] is programmed for SPI and used in Master mode, this bit is typically programmed as GPIO so it can be tied to the $\overline{\text{SPISEL}}$ pin of slave device. **Note:** I2C/SPI input pins are tied as deasserted when not selected.
27–31	—	Reserved

1 The functionality of GPIOCR[I2C2_SCL] is overridden if GPIOCR[UART0] is programmed for use with SPI.

2 The functionality of GPIOCR[I2C2_SDA], is affected by GPIOCR[UART0] in the following manner:
If GPIOCR[UART0]set for SPI and GPIOCR[I2C2_SDA] is cleared, the pin operates with $\overline{\text{SPISEL}}$ functionality
If GPIOCR[UART0]set for SPI and GPIOCR[I2C2_SDA] is set, the pin operates with GPIO2 functionality
If GPIOCR[UART0] not set for SPI, then GPIOCR[I2C2_SDA] acts as a select between GPIO2 and I2C2 functionality.

23.4.1.9 Alternate Function Signal Multiplex Control Register (PMUXCR)

Shown in Figure 23-9, PMUXCR contains bits that enable local bus parity, SSI, memory debug, and external DMA signaling which exist as alternate functions on eLBC pins, and interrupt input pins IRQ[6:11].

Offset 0x060 Access: Read/Write

	0												14	15
R W	—													LDPSEL
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23		27	28	29	30	31
R W	SSI1		SSI2		LA_22_25		DBG DRV		—		DMA2_0	DMA2_3	DMA1_0	DMA1_3
Reset	*n*	0	*n*	0	0	0	*n*	0	0	0	0	0	0	0

Figure 23-9. Alternate Function Pin Multiplex Control Register (PMUXCR)

Table 23-12 describes the bit settings of PMUXCR.

Table 23-12. PMUXCR Field Descriptions

Bits	Name	Description
0–14	—	Reserved
15	LDPSEL	Local bus data parity select. Enables data parity on the local bus. 0 LDP[0:3] used for latched address LA[6:9] **Note:** 1LDP[0:3] used for data parity LDP[0:3]
16–17	SSI1	Enables SSI1 interface. The initial value of the most significant bit is determined by PORDEVSR[LA_SEL]. 00 LA[10:15] used for latched address LA[10:15] 01 LA[10:15] not driven (high-impedance) 10 LA[10:15] used for SSI1 11 Reserved

Table 23-12. PMUXCR Field Descriptions (continued)

Bits	Name	Description
18–19	SSI2	Enables SSI2 interface. The initial value of the most significant bit is determined by PORDEVSR[LA_SEL]. 00 LA[16:21] used for latched address LA[16:21] 01 LA[16:21] not driven (high-impedance) 10 LA[16:21] used for SSI2 11 Reserved
20–21	LA_22_25	Disables LA[22:25] 00 LA[22:25] used for latched address LA[22:25] 01 LA[22:25] not driven (high-impedance) 10 Reserved 11 Reserved
22	DBGDRV	Debug drive. Enables the eLBC/DDR debug signals. See Section 25.1.3, "Modes of Operation," for correct usage of this bit. The initial value of this bit is determined by PORDBGMSR[MEM_SEL]. 0 MSRCID[0:4] and MDVAL signals are driven **Note:** 1MSRCID[0:4] and MDVAL signals aren't driven (high-impedance)
28	DMA2_0	Enables DMA controller 2, channel 0 signals for external control of DMA transfers through this channel. 0 DMA controller 2, channel 0 is not exposed to pins; the pins retain their primary function as eLBC signals. Note that in this case, DMA controller 2, channel 0 is still functional, but cannot perform externally controlled transfers. 1 DMA controller 2, channel 0 is exposed to pins as follows: $\overline{\text{LCS5}}$ functions as $\overline{\text{DMA2\_DREQ0}}$ $\overline{\text{LCS6}}$ functions as $\overline{\text{DMA2\_DACK0}}$ $\overline{\text{LCS7}}$ functions as $\overline{\text{DMA2\_DDONE0}}$
29	DMA2_3	Enables DMA controller 2, channel 3 signals for external control of DMA transfers through this channel. 0 DMA controller 2, channel 3 is not exposed to pins; the pins retain their primary function as eLBC signals. Note that in this case, DMA controller 2, channel 3 is still functional, but cannot perform externally controlled transfers. 1 DMA controller 2, channel 3 is exposed to pins as follows: GPIO2[29] functions as $\overline{\text{DMA2\_DREQ3}}$ GPIO2[30] functions as $\overline{\text{DMA2\_DACK3}}$ GPIO2[31] functions as $\overline{\text{DMA2\_DDONE3}}$

Table 23-12. PMUXCR Field Descriptions (continued)

Bits	Name	Description
30	DMA1_0	Enables DMA controller 1, channel 0 signals for external control of DMA transfers through this channel. 0 DMA controller 1, channel 0 is not exposed to pins; the pins retain their primary function as eLBC signals. Note that in this case, DMA controller 1, channel 0 is still functional, but cannot perform externally controlled transfers. 1 DMA controller 1, channel 0 is exposed to pins as follows: IRQ[6] functions as $\overline{\text{DMA1\_DREQ0}}$ IRQ[7] functions as $\overline{\text{DMA1\_DACK0}}$ IRQ[8] functions as $\overline{\text{DMA1\_DDONE0}}$
31	DMA1_3	Enables DMA controller 1, channel 3 signals for external control of DMA transfers through this channel. 0 DMA controller 1, channel 3 is not exposed to pins; the pins retain their primary function as eLBC signals. Note that in this case, DMA controller 1, channel 3 is still functional, but cannot perform externally controlled transfers. 1 DMA controller 1, channel 3 is exposed to pins as follows: IRQ[9] functions as $\overline{\text{DMA1\_DREQ3}}$ IRQ[10] functions as $\overline{\text{DMA1\_DACK3}}$ IRQ[11] functions as $\overline{\text{DMA1\_DDONE3}}$

23.4.1.10 Device Disable Register (DEVDISR)

DEVDISR and DEVDISR2 contain disable control bits for various functional blocks. When a block is disabled by the DEVDISR register, its clocks are shut down to save power. Note that a block can be enabled in DEVDISR and disabled functionally. The SerDes blocks are enabled or disabled based on the I/O ports POR config. See

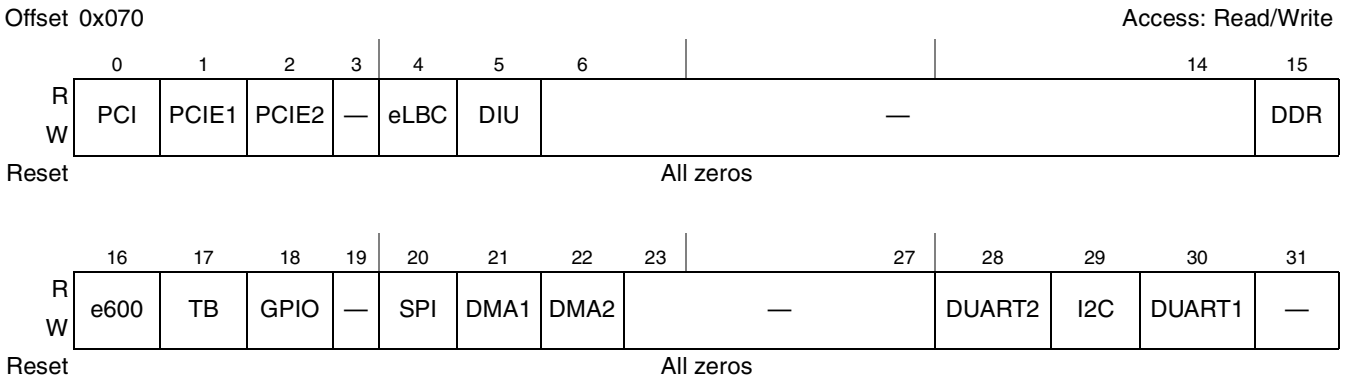


Figure 23-10. Device Disable Register (DEVDISR)

All functional blocks are enabled after reset; unneeded blocks can be disabled to reduce power consumption or allow their signals to be used as general-purpose I/O signals. Blocks disabled by DEVDISR must not be re-enabled without a hard reset. Table 23-13 describes DEVDISR fields.

Table 23-13. DEVDISR Field Descriptions

Bits	Name	Description
0	PCI	PCI controller disable 0 PCI controller is enabled (clocks are on). 1 PCI controller is disabled (clocks are shut down to minimize power).
1	PCIE1	PCI Express controller 1 (x4) disable 0 PCI Express controller 1 (x4) is enabled (clocks are on). 1 PCI Express controller 1 (x4) is disabled (clocks are shut down to minimize power).
2	PCIE2	PCI Express controller 2 (x8) disable 0 PCI Express controller 2 (x8) is enabled (clocks are on). 1 PCI Express controller 2 (x8) is disabled (clocks are shut down to minimize power).
3	—	Reserved
4	eLBC	Enhanced local bus controller disable 0 Enhanced local bus controller is enabled (clocks are on). 1 Enhanced local bus controller is disabled (clocks are shut down to minimize power).
5	DIU	Display interface unit (LCD controller) disable 0 Display interface unit is enabled (clocks are on). 1 Display interface unit is disabled (clocks are shut down to minimize power).
6–14	—	Reserved
15	DDR	DDR SDRAM controller disable 0 DDR SDRAM controller is enabled (clocks are on). 1 DDR SDRAM controller is disabled (clocks are shut down to minimize power).
16	e600	e600 core disable 0 e600 core is enabled 1 e600 core is disabled. The core in the stopped state, in which it does not respond to interrupts. Instruction fetching is stopped, snooping is disabled, PLL is disabled, and clocks are shut down to all functional units of the core.
17	TB	Time base (timer facilities) of the e600 core disable 0 If DEVDISR[e600] = 0, the timer facilities for e600 core are enabled. 1 The timer facilities for e600 core are disabled
18	GPIO	GPIO controller disabled 0 GPIO controller is enabled (clocks are on). 1 GPIO controller is disabled (clocks are shut down to minimize power).
19	—	Reserved
20	SPI	SPI controller disable 0 SPI controller is enabled (clocks are on). 1 SPI controller is disabled (clocks are shut down to minimize power).
21	DMA1	DMA controller 1 disable 0 DMA controller 1 is enabled (clocks are on). 1 DMA controller 1 is disabled (clocks are shut down to minimize power).

Table 23-13. DEVDISR Field Descriptions (continued)

Bits	Name	Description
22	DMA2	DMA controller 2 disable 0 DMA controller 2 is enabled (clocks are on). 1 DMA controller 2 is disabled (clocks are shut down to minimize power).
23–27	—	Reserved
28	DUART2	DUART controller 2 disabled (Step 2) 0 DUART controller 2 (both UARTs) are enabled (clocks are on). 1 DUART controller 2 (both UARTs) are disabled (clocks are shut down to minimize power).
29	I2C	I^2C controllers disabled (Step 2) 0 The I^2C controllers are enabled (clocks are on) 1 The I^2C controllers are disabled (clocks are shut down to minimize power).
30	DUART1	DUART controller 1 (both UARTs) disabled (Step 2) 0 DUART controller 1 (both UARTs) are enabled (clocks are on). 1 DUART controller 1 (both UARTs) are disabled (clocks are shut down to minimize power).
31	—	Reserved

23.4.1.11 Device Disable Register 2 (DEVDISR2)

Like DEVDISR, DEVDISR2 contains disable control bits for various functional blocks. When a block is disabled by theDEVDISR2 register, its clocks are shut down to save power. Note that a block can be enabled in DEVDISR2 and disabled functionally.

Offset 0x074 Access: Read/Write

	0	1	2	3	4	5	6						31
R W	IR1	IR2	GTM1	GTM2	SSI1	SSI2	—						
Reset	All zeros												

Figure 23-11. Device Disable Register 2 (DEVDISR2)

All functional blocks are enabled after reset; unneeded blocks can be disabled to reduce power consumption or allow their signals to be used as general-purpose I/O signals. Blocks disabled by DEVDISR2 must not be re-enabled without a hard reset. Table 23-14 describes DEVDISR2 fields.

Table 23-14. DEVDISR2 Field Descriptions

Bits	Name	Description
0	IR1	IR controller 1 disable (disables IR1's FIR and SIR controllers) 0 IR controller 1 is enabled (clocks are on). 1 IR controller 1 is disabled (clocks are shut down to minimize power).
1	IR2	IR controller 2 disable (disables IR2's FIR and SIR controllers) 0 IR controller 2 is enabled (clocks are on). 1 IR controller 2 is disabled (clocks are shut down to minimize power).
2	GTM1	Global timer module 1 disable 0 Global timer module 1 is enabled (clocks are on). 1 Global timer module 1 is disabled (clocks are shut down to minimize power).

Table 23-14. DEVDISR2 Field Descriptions (continued)

Bits	Name	Description
3	GTM2	Global timer module 2 disable 0 Global timer module 2 is enabled (clocks are on). 1 Global timer module 2 is disabled (clocks are shut down to minimize power).
4	SSI1	SSI1 disable 0 SSI1 is enabled (clocks are on). 1 SSI1 is disabled (clocks are shut down to minimize power).
5	SSI2	SSI2 disable 0 SSI2 is enabled (clocks are on). 1 SSI2 is disabled (clocks are shut down to minimize power).
6–31	—	Reserved

23.4.1.12 Power Management Control and Status Register (POWMGTCSR)

Shown in Figure 23-12, POWMGTCSR contains bits for placing the device into low power states and for controlling when it wakes up. It also contains power management status bits.

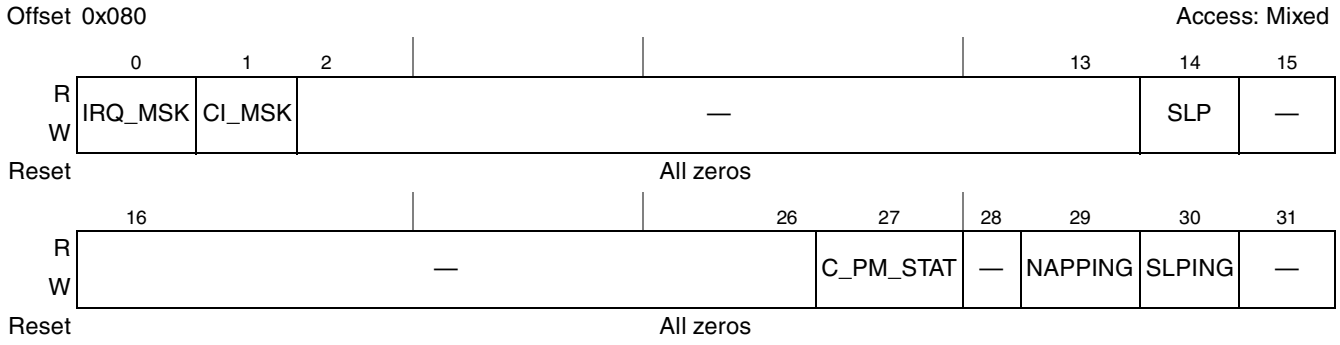


Figure 23-12. Power Management Control & Status Register (POWMGTCSR)

Table 23-15 describes the bit settings of PMUXCR.

Table 23-15. POWMGTCSR Field Descriptions

Bits	Name	Description
0	IRQ_MSK	e600 core Interrupt input mask 0 Interrupts cause e600 core to wake up from a low-power state. 1 Interrupts to e600 core are masked as a wake-up condition. e600 core remains in a low-power state despite the presence of an interrupt request.
1	CI_MSK	e600 core Critical interrupt input mask 0 Critical interrupts cause e600 core to wake up from a low power state. 1 Critical interrupts to e600 core are masked as a wake-up condition. e600 core remains in a low-power state despite the presence of a critical interrupt (machine check).
2–13	—	Reserved
14	SLP	Sleep mode 0 No request to put device in sleep mode. 1 Device is to be placed in sleep mode. Instruction fetching is halted, snooping of L1 caches is disabled, and most functional blocks are shut down in the e600 core and the system logic.

Table 23-15. POWMGTCSR Field Descriptions (continued)

Bits	Name	Description
15–26	—	Reserved
27	C_PM_STAT	e600 core power management status 0 e600 core is not in doze, nap, or sleep mode. 1 e600 core is in doze, nap, or sleep mode because HID0[NAP] or HID0[SLP], and MSR[POW] are set. The core has halted instruction fetching and all functional units (except timer facilities in doze and nap mode) are quiesced. Note: If this bit is set and the associated core napping or sleeping bit is not set, then the core is in dozing state. For e600, dozing is a transient state. See Section 23.5.1, "Power Management," for details.
28	—	Reserved
29	NAPPING	Nap status 0 The device is not in nap mode. 1 The device is in nap mode because HID0[NAP] and MSR[POW] in the e600 core are set and the platform has no pending snoop transactions. The core has halted instruction fetching, snooping to the L1 and L2 caches is disabled, and all of the core's functional units except the timer facilities are shut down. All functional blocks in the device are running (if not disabled).
30	SLPING	Sleep status 0 The device is not attempting to reach sleep mode. 1 The device is attempting to enter sleep mode because POWMGTCSR[SLP] is set, or HID0[SLEEP] and MSR[POW] in the e600 core are set. The core has halted fetching, snooping to the L1 and L2 caches is disabled, and all of the core's functional units, including the timer facilities, are shut down. Most functional blocks in device are shut down or are attempting to shut down.
31	—	Reserved

23.4.1.13 Machine Check Summary Register (MCPSUMR)

Shown in Figure 23-16, MCPSUMR contains bits summarizing some of the sources of a pending machine check interrupt. All MCPSUMR bits function as write-1-to-clear.

Table 23-16 describes the bit settings of MCPSUMR.

Table 23-16. MCPSUMR Field Descriptions

Bits	Name	Description
0–28	—	Reserved
29	WRS	Watchdog timer machine check 0 Machine check exception was not caused by watchdog timer. 1 Machine check exception was caused by a soft reset condition (i.e. watchdog programmed for non-maskable interrupt and timeout occurred).
30	SRESET	Soft reset 0 Soft reset exception was not caused by $\overline{sreset}$ assertion. 1 Reset exception was caused by the assertion of the $\overline{sreset}$ input signal.
31	MCP_IN	*Machine check* 0 Machine check exception was not caused by $\overline{mcp}$ assertion. 1 Machine check exception was caused by the assertion of the $\overline{mcp}$ input signal.

23.4.1.14 Reset Request Status and Control Register (RSTRSCR)

Shown in Figure 23-16, RSTRSCR contains bits summarizing some of the sources of a reset interrupt. All RSTRSCR bits are read-only.

Table 23-16 describes the bit settings of MCPSUMR.

Table 23-17. MCPSUMR Field Descriptions

Bits	Name	Description
0–9	—	Reserved
10	BS_RR	Boot sequencer reset request 0 Reset not requested by boot sequencer. 1 Reset requested by boot sequencer (due to error).
11	WDT_RR	Watchdog timer reset request 0 Reset not requested by watchdog timer. 1 Reset requested by watchdog timer (due to timeout).
12	SW_RR	Software settable reset request 0 Reset not requested by software settable reset request. 1 Reset requested by software settable reset request (due to setting RSTCR[HRESET_REQ]).
13–31	—	Reserved

23.4.1.15 Processor Version Register (PVR)

Shown in Figure 23-13, the PVR contains the e600 processor version number. It is a memory-mapped copy of the PVR from the e600 core (and is therefore accessible to external devices). See Section 6.1.4.1, “Processor Version Register (PVR),” for information on the e600 core PVR register.

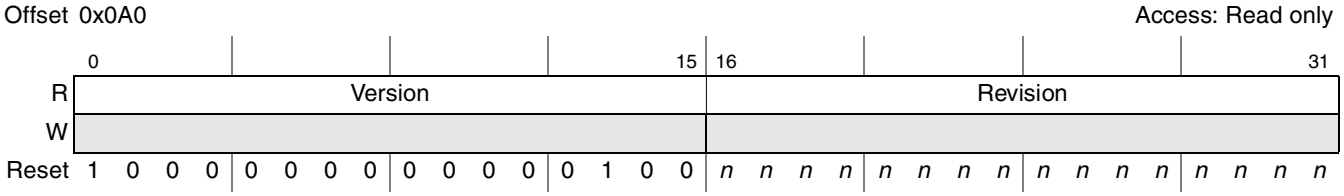


Figure 23-13. Processor Version Register (PVR)

Table 23-18 describes the fields of PVR.

Table 23-18. PVR Field Descriptions

Bits	Name	Description
0–15	Version	A 16-bit number that identifies the version of the processor core. Different version numbers indicate major differences between processors, such as which optional facilities and instructions are supported.
16–31	Revision	A 16-bit number that distinguishes between implementations of the version. Different revision numbers indicate minor differences between processors having the same version number, such as clock rate and engineering change level.

23.4.1.16 System Version Register (SVR)

Shown in Figure 23-14, the SVR contains the system version number for the MPC8610 implementation. This value can also be read though the SVR SPR of the e600 core. See Section 6.1.4.2, "System Version Register (SVR)," for information on the e600 core SVR register.

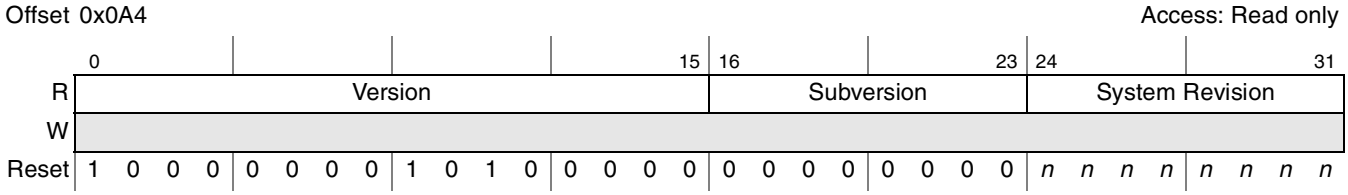


Figure 23-14. System Version Register (SVR)

Table 23-19 describes the fields of SVR.

Table 23-19. SVR Field Descriptions

Bits	Name	Description
0–15	Version	A 16-bit number that identifies the version of the device. Different version numbers indicate major differences between devices, such as which optional facilities and instructions are supported.
16–23	Subversion	An 8-bit number that further differentiates the device.
24–31	Revision	System revision for the MPC8610 system logic.

23.4.1.17 Reset Control Register (RSTCR)

Shown in Figure 23-15, the RSTCR contains the reset control bits. Through this register, software can request the assertion of device $\overline{\text{HRESET\_REQ}}$.

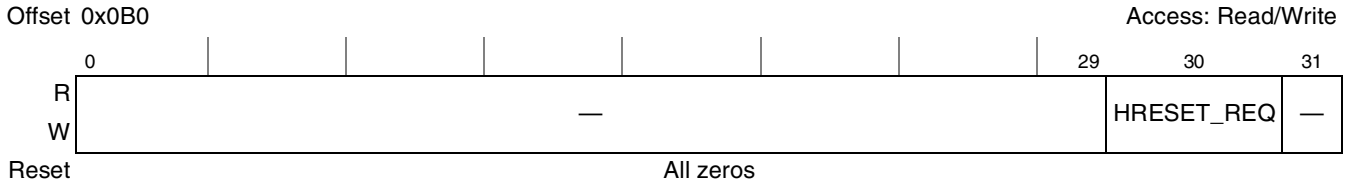


Figure 23-15. Reset Control Register (RSTCR)

Table 23-20 describes the bit settings of RSTCR.

Table 23-20. RSTCR Field Descriptions

Bits	Name	Description
0–29	—	Reserved
30	HRESET_REQ	Hardware reset request 0 No hardware reset request 1 Request hardware reset
31	—	Reserved

23.4.1.18 eLBC Voltage Select Control Register (eLBCVSELCR)

Shown in Figure 23-15, the eLBCVSELCR contains the eLBC voltage control bits. Through this register, software can select the I/O voltage level for the enhanced local bus controller (eLBC).

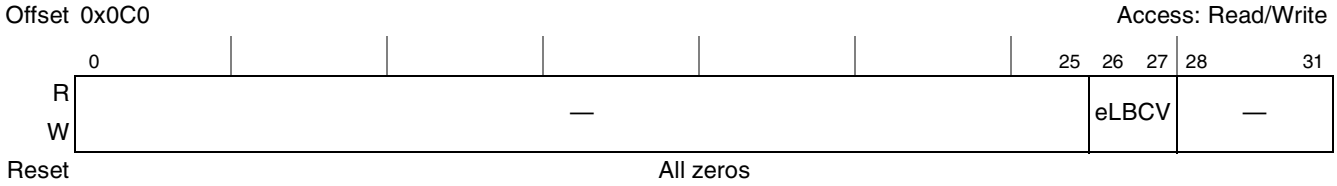


Figure 23-16. eLBC Voltage Select Control Register (eLBCVSELCR)

Table 23-20 describes the bit settings of eLBCVSELCR.

Table 23-21. eLBCVSELCR Field Descriptions

Bits	Name	Description
0–25	—	Reserved
26–27	eLBCV	eLBC I/O voltage level (nominal) 00 3.3 V (default) 01 2.5 V 10 1.8 V 11 Reserved
28–31	—	Reserved

23.4.1.19 Clock Divide Register (CLKDVDR)

Shown in Figure 23-17, CLKDVDR contains bits for controlling the DIU pixel clock and the SSI interface clock.

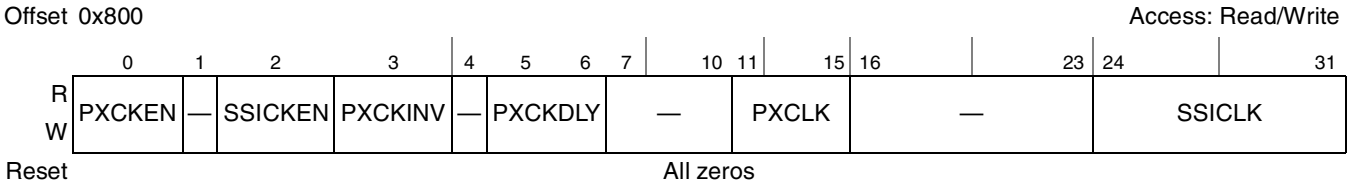


Figure 23-17. Clock Divide Register (CLKDVDR)

Table 23-22 describes the bit settings of CLKDVDR.

Table 23-22. CLKDVDR Field Descriptions

Bits	Name	Description
0	PXCKEN	Pixel clock enable 0 The pixel clock is disabled. 1 The pixel clock is enabled.
1	—	Reserved

Table 23-22. CLKDVDR Field Descriptions (continued)

Bits	Name	Description
2	SSICKEN	SSI clock enable. See Section 16.4.1, "SSI Clocking," for more information. 0 The SSI clock is disabled. 1 The SSI clock is enabled.
3	PXCKINV	Pixel clock invert 0 The pixel clock is not inverted. 1 The pixel clock is inverted. **Note:** The pixel clock must be disabled (PXCKEN = 0) before modifying this field.
4	—	Reserved
5–6	PXCKDLY	Pixel clock delay 00 The pixel clock is not delayed. 01 The pixel clock is delayed 1 platform clock. 10 The pixel clock is delayed 2 platform clocks. 11 The pixel clock is delayed 3 platform clocks. **Note:** The pixel clock must be disabled (PXCKEN = 0) before modifying this field.
7–10	—	Reserved
11–15	PXCLK	Pixel clock. In general, the pixel clock rate is set by Platform Clock ÷ (PXCLK + 1). The pixel clock has a 50% duty cycle regardless of the divisor. 00000 Reserved 00001 Reserved 00010 Platform clock ÷ 3 00011 Platform clock ÷ 4 ... 11100 Platform clock ÷ 29 11101 Platform clock ÷ 30 11110 Platform clock ÷ 31 11111 Platform clock ÷ 32 **Note:** The pixel clock must be disabled (PXCKEN = 0) before modifying this field.
16–23	—	Reserved
24–31	SSICLK	SSI clock. In general, the SSI controller clock rate is set by platform clock ÷ (SSICLK + 1). See Section 16.4.1, "SSI Clocking," for more information. 00000000 Reserved 00000001 Platform clock ÷ 2 00000010 Platform clock ÷ 3 00000011 Platform clock ÷ 4 ... 11111100 Platform clock ÷ 253 11111101 Platform clock ÷ 254 11111110 Platform clock ÷ 255 11111111 Platform clock ÷ 256 **Note:** SSICLK must not be modified when the SSI block is enabled (SCR[SSIEN] = 1) nor when the SSI clock is enabled (CLKDVDR[SSICKEN] = 1).

23.4.1.20 InfraRed Control Register (IRCR)

Shown in Figure 23-18, the IRCR controls the configuration of the IR interfaces.

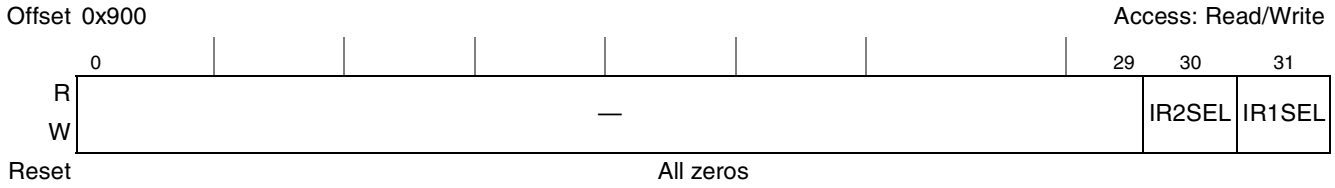


Figure 23-18. InfraRed Control Register (IRCR)

Table 23-23 describes the bit settings of IRCR.

Table 23-23. IRCR Field Descriptions

Bits	Name	Description
0–29	—	Reserved
30	IR2SEL	IR2 mode select. This bit selects the multiplexer option for routing the TX outputs of either the SIRI or FIRI controller to the external IR2_TXD signals. 0 IR2 configured as SIRI 1 IR2 configured as FIRI
31	IR1SEL	IR1 mode select. This bit selects the multiplexer option for routing the TX outputs of either the SIRI or FIRI controller to the external IR1_TXD signals. 0 IR1 configured as SIRI 1 IR1 configured as FIRI

23.4.1.21 MCM Control Register (MCMCR)

Shown in Figure 23-19, MCMCR contains bits for controlling DIU initiated transactions routed through the MCM.

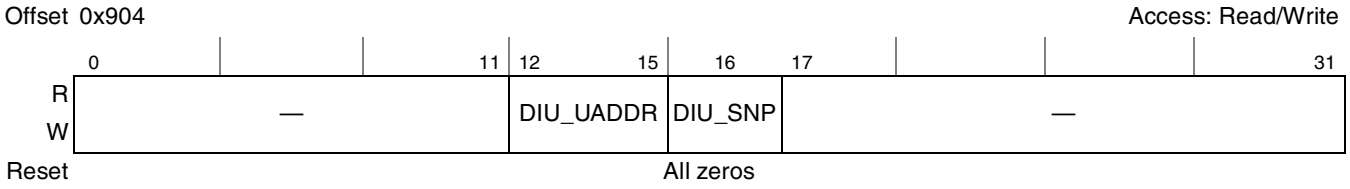


Figure 23-19. MCM Control Register (MCMCR)

Table 23-24 describes the bit settings of MCMCR.

Table 23-24. MCMCR Field Descriptions

Bits	Name	Description
0–11	—	Reserved
12–15	DIU_UADDR	DIU upper address. These bits provide the highest order address bits for all tranactions initiated by the DIU.

Table 23-24. MCMCR Field Descriptions (continued)

Bits	Name	Description
16	DIUSNP	DIU transaction snoop disable 0 DIU transactions are snooped 1 DIU transactions are not snooped.
17–31	—	Reserved

23.4.1.22 DMA Control Register (DMACR)

Shown in Figure 23-20, DMACR contains bits for assigning the DMA controller channels to the SSI or InfraRed interfaces.

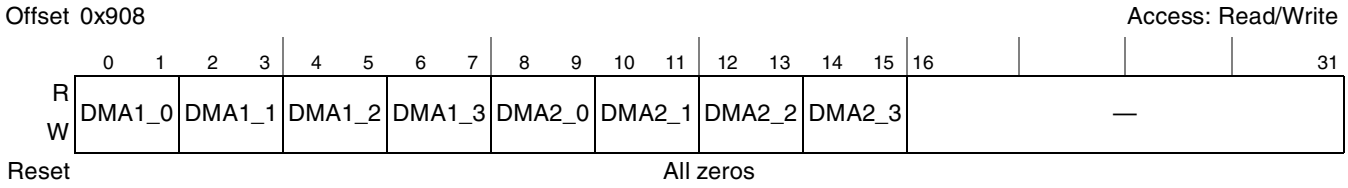


Figure 23-20. DMA Control Register (MCMCR)

Table 23-25 describes the bit settings of DMACR.

Table 23-25. DMACR Field Descriptions

Bits	Name	Description
0–1	DMA1_0	DMA controller 1, channel 0 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved **Note:** This field is ignored if PMUXCR[DMA1_0] is set for DMA operation.
2–3	DMA1_1	DMA controller 1, channel 1 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved
4–5	DMA1_2	DMA controller 1, channel 2 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved
6–7	DMA1_3	DMA controller 1, channel 3 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved **Note:** This field is ignored if PMUXCR[DMA1_3] is set for DMA operation.

Table 23-25. DMACR Field Descriptions (continued)

Bits	Name	Description
8–9	DMA2_0	DMA controller 2, channel 0 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved **Note:** This field is ignored if PMUXCR[DMA2_0] is set for DMA operation.
10–11	DMA2_1	DMA controller 2, channel 1 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved
12–13	DMA2_2	DMA controller 2, channel 2 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved
14–15	DMA2_3	DMA controller 2, channel 3 transfers may be initiated by: 00 SSI controllers 01 IR controllers 10 Reserved 11 Reserved **Note:** This field is ignored if PMUXCR[DMA2_3] is set for DMA operation.
16–31	—	Reserved

23.4.1.23 eLBC Control Register (ELBCCR)

Shown in Figure 23-21, the ELBCCR contains parameters for controlling a frequency prescaler and the LCK*n* output pins.

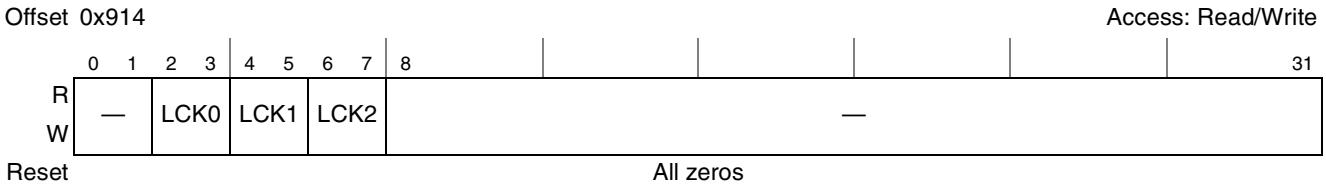


Figure 23-21. eLBC Control Register (ELBCCR)

Table 23-26 describes the bit settings of ELBCCR.

Table 23-26. ELBCCR Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–3	LCK0	LCK0 clock 00 LCK0 operates normally. 01 LCK0 high-impedance 10 Reserved 11 Reserved

Table 23-26. ELBCCR Field Descriptions (continued)

Bits	Name	Description
4–5	LCK1	LCK1 clock 00 LCK1 operates normally. 01 LCK1 high-impedance 10 Reserved 11 Reserved
6–7	LCK2	LCK2 clock 00 LCK2 operates normally. 01 LCK2 high-impedance 10 Reserved 11 Reserved
8–31	—	Reserved

23.4.1.24 DDR Clock Disable Register (DDRCLKDR)

Shown in Figure 23-22, DDRCLKDR contains parameters for controlling the driver characteristics of the DDR controller.

Offset 0xB28 Access: Read/Write

	0	25	26	27	28	29	30	31
R W	—		MCK0_DIS	MCK1_DIS	MCK2_DIS	MCK3_DIS	MCK4_DIS	MCK5_DIS
Reset	All zeros							

Figure 23-22. DDR Clock Disable Register (DDRCLKDR)

Table 23-27 describes the bit settings of DDRCLKDR.

Table 23-27. DDRCLKDR Field Descriptions

Bits	Name	Description
0–25	—	Reserved
26	MCK0_DIS	MCK0 disable 0 MCK0 is enabled. 1 MCK0 is disabled.
27	MCK1_DIS	MCK1 disable 0 MCK1 is enabled. 1 MCK1 is disabled.
28	MCK2_DIS	MCK2 disable 0 MCK2 is enabled. 1 MCK2 is disabled.
29	MCK3_DIS	MCK3 disable 0 MCK3 is enabled. 1 MCK3 is disabled.

Table 23-27. DDRCLKDR Field Descriptions (continued)

Bits	Name	Description
30	MCK4_DIS	MCK4 disable 0 MCK4 is enabled. 1 MCK4 is disabled.
31	MCK5_DIS	MCK5 disable 0 MCK5 is enabled. 1 MCK5 is disabled.

23.4.1.25 CLK_OUT Control Register (CLKOCR)

Shown in Figure 23-23, the CLKOCR contains control bits that select the clock sources to be placed on the clock out (CLK_OUT) signal.

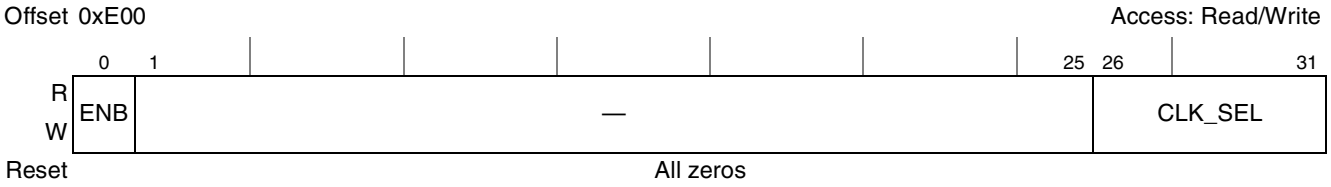


Figure 23-23. Clock Out Control Register (CLKOCR)

Table 23-28 describes the bit settings of CLKOCR.

Table 23-28. CLKOCR Field Descriptions

Bits	Name	Description
0	ENB	Clock out enable 0 CLK_OUT signal is tri-stated 1 CLK_OUT signal is driven by the source selected by CLKOCR[CLK_SEL]
1–25	—	Reserved
26–31	CLK_SEL	Clock out select 000000 Platform Clock (*plat_clk*, also called MPX_clk) 000001 Platform Clock ÷ 2 000010 SYSCLK (echoes SYSCLK input) 000011 SYSCLK ÷ 2 (demonstrates platform PLL lock) 10*n*010 PCI bus clock 10*n*011 PCI bus clock ÷ 2 All others reserved.

23.4.1.26 SerDes 1 Debug Control Register 0 (SRDS1DCR0)

Shown in Figure 23-24, the SRDS1DCR0 contains control bits for the ×4 SerDes on high speed interface port 1.

Offset 0xF04 Access: Read/Write

	0–15	16–19	20–23	24–31
R W	—	XMITEQAD	XMITEQEH	—
Reset	0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1	0 1 0 0	0 1 0 0	0 0 0 1 1 0 0 0

Figure 23-24. SerDes 1 Debug Control Register 0 (SRDS1DCR0)

Table 23-29 describes the bit settings of SRDS1DCR0.

Table 23-29. SRDS1DCR0 Field Descriptions

Bits	Name	Description
0–15	—	Reserved. Software must preserve the values of these bits.
16–19	XMITEQAD	Transmit equalization selection bus for SD1 lanes 0-1 (a-b) Default value: 0100 (PCI Express) Bit 0 is amplitude select 0 5/6 Vdd-diff-pk=pk 1 Vdd-diff-pk=pk Bits 1:3 are equalization amplitude: 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
20–23	XMITEQEH	Transmit equalization selection bus for SD1 lanes 2-3 (e-f) Default value: 1100 (PCI Express) Bit 0 is amplitude select 0 5/6 Vdd-diff-pk=pk 1 Vdd-diff-pk=pk Bits 1:3 are defined: 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
24–31	—	Reserved. Software must preserve the values of these bits.

23.4.1.27 SerDes 1 Debug Control Register 1 (SRDS1DCR1)

Shown in Figure 23-25, SRDS1DCR1 contains control bits for the ×4 SerDes on high speed interface port 1.

Offset 0xF08 Access: Read/Write

	0	1	2	3	4	5	6																					31
R W	PD0	PD1	—		PD2	PD3	—																					
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	*n*	0	0	0

Figure 23-25. SerDes 1 Debug Control Register 1 (SRDS1DCR1)

Table 23-30 describes the bit settings of SRDS1DCR1.

Table 23-30. SRDS1DCR1 Field Descriptions

Bits	Name	Description
0	PD0	Lane 0 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
1	PD1	Lane 1 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
2–3	—	Reserved
4	PD2	Lane 2 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
5	PD3	Lane 3 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
6–31	—	Reserved

23.4.1.28 SerDes 2 Debug Control Register 0 (SRDS2DCR0)

Shown in Figure 23-26, the SRDS2DCR0 contains control bits for the ×8 SerDes on high speed interface port 2.

Offset 0xF40 Access: Read/Write

	0															15	16			19	20			23	24							31
R W	—																XMITEQAD				XMITEQEH				—							
Reset	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	1	1	0	0	0

Figure 23-26. SerDes 2 Debug Control Register 0 (SRDS2DCR0)

Table 23-31 describes the bit settings of SRDS2DCR0.

Table 23-31. SRDS2DCR0 Field Descriptions

Bits	Name	Description
0-15	—	Reserved. Software must preserve the values of these bits.
16–19	XMITEQAD	Transmit equalization selection bus for SD2 lanes 0-3 (a-d) Default value: 0100 (PCI Express) Bit 0 is amplitude select 0 5/6 Vdd-diff-pk=pk 1 Vdd-diff-pk=pk Bits 1:3 are equalization amplitude: 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
20–23	XMITEQEH	Transmit equalization selection bus for SD2 lanes 4-7 (e-h) Default value: 1100 (PCI Express) Bit 0 is amplitude select 0 5/6 Vdd-diff-pk=pk 1 Vdd-diff-pk=pk Bits 1:3 are defined: 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
24–31	—	Reserved. Software must preserve the values of these bits.

23.4.1.29 SerDes 2 Debug Control Register 1 (SRDS2DCR1)

Shown in Figure 23-27, SRDS2DCR1 contains control bits for the ×8 SerDes on high speed interface port 1.

Offset 0xF44 Access: Read/Write

	0	1	2	3	4	5	6	7	8																31
R W	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7	—																
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	*n*	0	0	0

Figure 23-27. SerDes 2 Debug Control Register 1 (SRDS2DCR1)

Table 23-32 describes the bit settings of SRDS2DCR1.

Table 23-32. SRDS1DCR1 Field Descriptions

Bits	Name	Description
0	PD0	Lane 0 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
1	PD1	Lane 1 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
2	PD2	Lane 2 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
3	PD3	Lane 3 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
4	PD4	Lane 4 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
5	PD5	Lane 5 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
6	PD6	Lane 6 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
7	PD7	Lane 7 power down 0 Normal 1 The serial links are disabled and power usage is minimized in all internal cells.
8–31	—	Reserved

23.5 Functional Description

23.5.1 Power Management

The MPC8610 has features to minimize power consumption at several levels. Software can shut down clocks to individual blocks when they are not needed through memory-mapped registers (DEVDISR and DEVDISR2). Additionally, software running on the e600 core can access the core's SPRs to put the device into nap or sleep power down state. Finally, software can access a memory-mapped register (POWMGTCR) in the global utilities block to put the device in a sleep state.

Note that the software that writes to either DEVDISR, DEVDISR2, or POWMGTCR can be running either on the e600 core or on an external master that can write to the MPC8610 memory-mapped registers through the PCI or PCI Express interfaces.

These features are described in further detail in this section.

23.5.1.1 Relationship Between Core and Device Power Management States

The MPC8610 has one low-power device state: sleep. The e600 core has three low-power states independent of device state: doze, nap, and sleep. The mapping of core power management states is shown in Figure 23-28 showing state transitions and platform controls from the perspective of the e600 core. The platform *qack* signal is based on pending snoops from a coherent I/O transaction.

Figure 23-28. e600 Core Power Management State Diagram

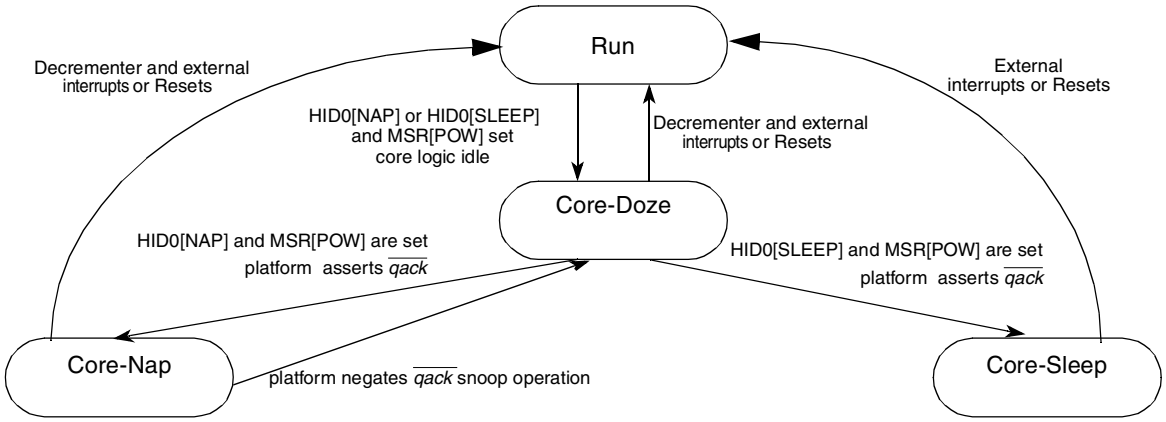


Table 23-33 lists basic characteristics of the low-power modes and the full on mode, along with the corresponding states of the MPC8610. Note that there are many other variables that control the state transitions between MPC8610 power management states.

Table 23-33. MPC8610 Power Management Modes—Basic Description

MPC8610 Mode	e600 Core Mode	Description	Core Responds To		Core Clocks
			Snoop	Interrupts	
Run	Full On	All units operating normally.	Yes	Yes	On
	Doze	Core is attempting to enter Nap or Sleep state. Core stops dispatching new instructions (core is halted). Platform automatically transitions the core from Doze to Nap or Sleep state when core interface is quiesced.	Yes	Yes	On
	Nap	Core is stopped with clocks off except to time base and decrementer. Platform automatically transitions the core from Nap to Doze state in order to service a snoop.	No	Yes	Dec/Time Base clock on & others off
	Sleep	Core is stopped with clocks off. Clocks powered down to all blocks (including core time base and decrementer).	No	Yes	PLL on & internal clocks off
Sleep	Sleep	Core must be attempting to enter Sleep state before POWMGTCSR[SLP] is set.	No	No	PLL on & internal clocks off

23.5.1.2 Shutting Down Unused Blocks

As described in Section 23.4.1.10, “Device Disable Register (DEVDISR),” and Section 23.4.1.11, “Device Disable Register 2 (DEVDISR2),” DEVDISR and DEVDISR2 provide a way to shut down certain functional blocks within the MPC8610 when they are not needed in a particular system. DEVDISR and DEVDISR2 can be written by the e600 core or by an external master. Powering down a block in this way turns off all clocks to that block.

With the exception of the timebase disable (TB) in DEVDISR, the controls in DEVDISR and DEVDISR2 were designed with the expectation that, once initialized by software, they would be modified only by a hard system reset ($\overline{\text{HRESET}}$). It is recommended that these registers be written only during system initialization. The results of re-enabling previously disabled blocks (by clearing the corresponding DEVDISR or DEVDISR2 field) without a hard reset are boundedly undefined. DEVDISR[TB] may be set and cleared without requiring a hard-reset. However, because changing other bits in DEVDISR could cause errant behavior, it is recommended that DEVDISR be accessed using a read-modify-write operation when changing TB.

NOTE

Functional blocks disabled using DEVDISR and DEVDISR2 cannot respond to configuration accesses. Any access to configuration, control, and status registers of a disabled block is a programming error.

23.5.1.3 Software-Controlled e600 Core Power-Down States

The e600 software can attempt to place the core in nap or sleep power-down states by writing to HID0 in the core. The device controls shutting off the clocks to the e600 core based on pending coherency transactions (snoops).

23.5.1.3.1 Doze Mode

In e600, doze mode is an intermediate state between full on or nap or sleep state. The e600 core remains in doze state until all pending snoops in the device are serviced.

In doze mode, the e600 core suspends instruction execution, significantly reducing the power consumption of the core. Snooping of the L1 and L2 caches is still supported and thus the data in the data cache is kept coherent. Interrupts directed to the core as described in Section 11.1.2, “Interrupts to the Processor Core,” are monitored by the device and cause the MPC8610 to use the defined handshake mechanism to exit the core from doze mode to allow the core to recognize and process the interrupt. Note that on any interrupt or exception, the e600 core clears the bit associated with POW in SRR1[13]. So the core does not automatically return to the previous low power mode when it returns from the interrupt.

The e600 core’s timer facilities are still enabled during doze mode, and core time base interrupts can be generated. All device logic external to the core remains fully operational in doze mode.

23.5.1.3.2 Nap Mode

In nap mode all clocks internal to the e600 core are turned off except for its timer facilities clock (the core time base and decrementer). The L1 and L2 caches do not respond to snoops in nap mode, however a pending snoop will cause the device to transition the e600 core from nap to doze state, in which the snoop

can be serviced. Once the snoop processing is complete, the device transitions the e600 core back into nap state.

Similar to doze mode, interrupts occurring in nap mode cause the device to wake up the e600 core in order to service the interrupt. Note that the e600 does not automatically return to the previous low power mode because the e600 clears the bit associated with MSR[POW] in SRR1[13] upon taking an exception.

All device logic external to the e600 core remains fully operational in nap mode.

23.5.1.3.3 Core Sleep Mode

In core sleep mode all clocks internal to the e600 core are turned off including its timer facilities clock (the core time base and decrementer). The L1 and L2 caches do not respond to snoops in sleep mode, so software must flush the caches before entering sleep mode.

Similar to doze and nap modes, interrupts occurring in sleep mode cause the device to wake up the e600 core in order to service the interrupt. Note that the e600 does not automatically return to the previous low power mode because the e600 clears the bit associated with MSR[POW] in SRR1[13] upon taking an exception.

All device logic external to the e600 core remains fully operational in core sleep mode.

23.5.1.4 Software-Controlled Device Power-Down State

The e600 software or an external device can attempt to place the device in sleep power-down states by writing to the POWMGTCSR. Before entering device sleep mode, software must place the core into core sleep mode. For e600 software placing the device in sleep power-down state, the suggested code sequence is:

```
mtspr setting HID0[SLEEP] to 1
cache flushes etc.
store setting POWMGTCSR[SLP] to 1
sync
mtspr setting MSR[POW] to 1
```

23.5.1.4.1 Device Sleep Mode

In device sleep mode, the e600 core is in core sleep mode and all I/O interfaces in the device logic are shut down. Only the clocks to the MPC8610 PIC are still running so that an external interrupt can wake up the device. After the core and I/O interfaces have shut down, ASLEEP is asserted and READY is negated.

NOTE

Only external interrupts can wake the MPC8610 from sleep mode. Internal interrupt sources depend on an active clock for their operation and these are disabled in sleep mode.

23.5.1.5 Power Management Control Fields

The e600 core provides the following fields to signal power management requests to the MPC8610 device logic.

- MSR[POW]—Enables core power management modes selected by HID0[NAP,SLEEP].
- HID0[NAP]—Signals the MPC8610 to initiate nap mode.
- HID0[SLEEP]—Signals the MPC8610 to initiate core sleep mode.

These register fields and their functional relationship are shown in Figure 23-28. The *e600 Reference Manual* has details on accessing these power management control bits.

Once the e600 core is in core sleep mode, an external master can initiate device power management requests by setting or SLP bit in the memory-mapped power management control and status register (POWMGTCSR). Maintaining cache coherency requires significant preparation by the core before entering sleep mode. For this reason only the core can initiate core sleep mode.

23.5.1.6 Interrupts and Power Management

23.5.1.6.1 Interrupts and Power Management Controlled by MSR and HID0

When an interrupt is asserted to the CPU, the core saves portions of the MSR to SRR1, and restores those values on return from the routine. However, MSR[POW], which gates the *HID0[NAP] and HID0[SLEEP] bits*, is always cleared when saved to SRR1 and remains cleared when it is restored; hence the corresponding power management status bits POWMGTCSR[NAPPING,SLPING]) negate when the interrupt begins processing in the core. They do not return to their previous state when the core executes an **rfi** instruction.

23.5.1.6.2 Interrupts and Power Management Controlled by POWMGTCSR

The IRQ_MSK and CI_MSK fields of the POWMGTCSR register prevent interrupts or critical interrupts from waking the device from a low power state.

Note that interrupts caused by the system management ($\overline{\text{SMI}}$) and machine check ($\overline{\text{MCP}}$) signals are not masked by the IRQ_MSK and CI_MSK fields. See Section 23.4.1.12, “Power Management Control and Status Register (POWMGTCSR),” for detailed information about the bits of POWMGTCSR.

Note also that unmasked interrupts that occur while the device is in the process of going into the sleep state (before sleep is completely attained) can also cause the device to return the core to full power operation.

23.5.1.7 Requirements for Reaching and Recovering from Device Sleep State

In order to successfully reach the sleep state, I/O traffic to the device must be stopped. The logic that controls the power down sequence waits for all I/O interfaces to become idle. In some applications this may happen eventually without actively shutting down interfaces, but most likely, software will have to take steps to shut down the PCI Express and PCI interfaces before issuing the command (writing to POWMGTCSR[SLP]) to put the device into sleep state.

The PCI and PCI Express interfaces will begin retrying inbound transactions before entering a power down state. These interfaces could potentially be in an unknown state when they exit sleep if it was in the middle of a retry sequence when its internal clocks were shut down. Therefore it is strongly recommended that system software clear the memory space bit in the PCI and PCI Express Bus Command Registers before

putting the device in sleep mode. Software may also need to set the Agent Config Lock bit of the PCI Bus Function Register so that the device will not respond to configuration transactions.

Chapter 24
Device Performance Monitor

This chapter describes the performance monitor facility, which can be used to monitor and optimize performance. The e600 core implements a separate performance monitor for strictly core-related behavior, such as instruction timing and L1 cache operations. This is described in the *PowerPC e600 Core Reference Manual* (Freescale Document Order No. E600CORERM).

Section 24.4.7, "Performance Monitor Events," briefly describes the events that can be monitored. Refer to the individual chapters for a better understanding of these events.

24.1 Introduction

The MPC8610 includes a performance monitor facility that can be used to monitor and record selected behaviors of the integrated device. Although the performance monitor described here is similar in many respects to the performance monitor facility implemented on the e600 core, it differs in that it is implemented using memory-mapped registers and it counts events outside the e600 core, for example, PCI, DDR, and L2 cache events.

Performance monitor counters (PMC0–PMC9) are used to count events selected by the performance monitor local control registers. PMC0 is a 64-bit counter specifically designated to count cycles. PMC1–PMC9 are 32-bit counters that can monitor 64 counter-specific events in addition to counting 64 reference events.

The benefits of the on-chip performance monitor are numerous, and include the following:

- Because some systems or software environments are not easily characterized by signal traces or benchmarks, the performance monitor can be used to understand the MPC8610's behavior in any system or software environment.
- The performance monitor facility can be used to aid system developers when bringing up and debugging systems.
- System performance can be increased by monitoring memory hierarchy behavior. This can help to optimize algorithms used to schedule or partition tasks and to refine the data structures and distribution used by each task.

24.1.1 Overview

Figure 24-1 is a high-level block diagram of the performance monitor, which consists of a global control register (PMGC0), one 64-bit counter (PMC0), nine 32-bit counters, and two control registers per counter (18 total control registers). The global control register PMGC0 affects all counters and takes priority over local control registers. The local control registers are divided into two groups, as follows:

- Local control A registers control counter freezing, overflow condition enable, event selection, and burstiness. Local control register PMLCA0, which controls counter PMC0, does not contain event selection because PMC0 counts only cycles.
- Local control B registers control the start and stop triggering, contain the counters' threshold values, and the value of the threshold multiplier. Local control register PMLCB0, which controls PMC0, does not contain threshold information because PMC0 only counts cycles.

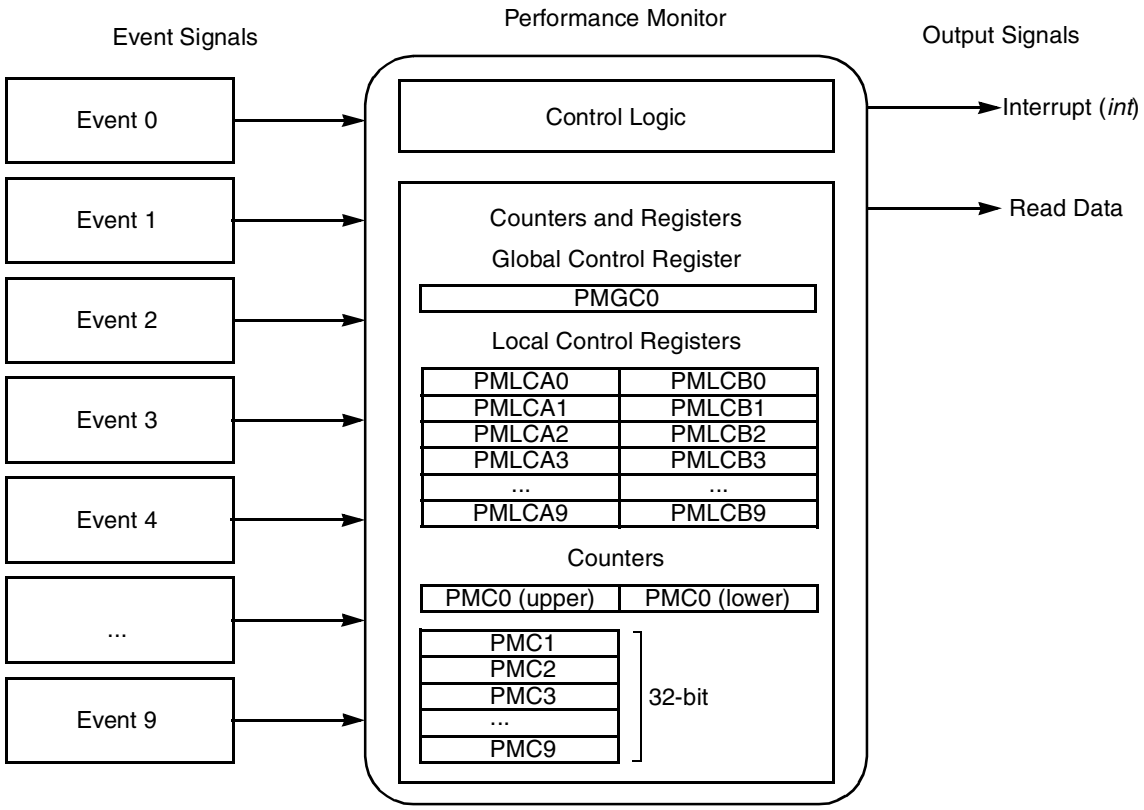


Figure 24-1. Performance Monitor Block Diagram

Performance monitor events are signalled by the functional blocks in the integrated device and are selectively recorded in the PMCs. Sixty-four of these events are referred to as reference events, which can be counted on any of the nine counters. Counter-specific events can be counted only on the counter where the event is defined.

The performance monitor can generate an interrupt on overflow. Several control registers specify how a performance monitor interrupt is signalled. The PMCs can also be programmed to freeze when an interrupt is signalled.

24.1.2 Features

The performance monitor offers a rich set of features that permits a complete performance characterization of the implementation. These features include:

- One 64-bit counter exclusively dedicated to counting cycles
- Nine 32-bit counters that count the occurrence of selected events

- One global control register (affects all counters) and two local control registers per counter
- Ability to count up to 64 reference events that may be counted on any of the nine 32-bit counters
- Ability to count up to 576 counter-specific events
- Triggering and chaining capability
- Duration threshold counting
- Burstiness feature that permits counting of burst events with a programmable time between bursts
- Ability to generate an interrupt on overflow

24.2 Signal Descriptions

The performance monitor does not have any signals that are driven externally (off-chip) but it does assert the internal interrupt (*int*) signal on a performance monitor interrupt condition.

24.3 Memory Map and Register Definition

The performance monitor uses nine counter registers and a group of local control registers that are used to specify the method of counting. Two local control registers are associated with each counter in addition to a global control register that applies to all counters.

Performance monitor registers reside in the CCSR space starting at block base address 0xE_1000. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved. This section describes the registers implemented to support the performance monitor facilities. Table 24-1 lists the performance monitor registers. These registers can be read or written only with 32-bit accesses.

Table 24-1. Control Register Memory Map

Address Offset (in Hex)	Register	Access	Reset	Section/Page
Performance Monitor Registers—Block Base Address 0xE_1000				
0x000	PMGC0—Performance monitor global control register	R/W	All zeros	24.3.1.1/24-5
0x010	PMLCA0—Performance monitor local control register A0	R/W	All zeros	24.3.1.2/24-5
0x014	PMLCB0—Performance monitor local control register B0	R/W	All zeros	24.3.1.2/24-5
0x018	PMC0 (lower)—Performance monitor counter 0 lower	R/W	All zeros	24.3.2.1/24-9
0x01C	PMC0 (upper)—Performance monitor counter 0 upper	R/W	All zeros	24.3.2.1/24-9
0x020	PMLCA1—Performance monitor local control register A1	R/W	All zeros	24.3.1.2/24-5
0x024	PMLCB1—Performance monitor local control register B1	R/W	All zeros	24.3.1.2/24-5
0x028	PMC1—Performance monitor counter 1	R/W	All zeros	24.3.2.1/24-9
0x030	PMLCA2—Performance monitor local control register A2	R/W	All zeros	24.3.1.2/24-5
0x034	PMLCB2—Performance monitor local control register B 2	R/W	All zeros	24.3.1.2/24-5
0x038	PMC2—Performance monitor counter 2	R/W	All zeros	24.3.2.1/24-9
0x040	PMLCA3—Performance monitor local control register A3	R/W	All zeros	24.3.1.2/24-5

Table 24-1. Control Register Memory Map (continued)

Address Offset (in Hex)	Register	Access	Reset	Section/Page
0x044	PMLCB3—Performance monitor local control register B3	R/W	All zeros	24.3.1.2/24-5
0x048	PMC3—Performance monitor counter 3	R/W	All zeros	24.3.2.1/24-9
0x050	PMLCA4—Performance monitor local control register A4	R/W	All zeros	24.3.1.2/24-5
0x054	PMLCB4—Performance monitor local control register B4	R/W	All zeros	24.3.1.2/24-5
0x058	PMC4—Performance monitor counter 4	R/W	All zeros	24.3.2.1/24-9
0x060	PMLCA5—Performance monitor local control register A5	R/W	All zeros	24.3.1.2/24-5
0x064	PMLCB5—Performance monitor local control register B 5	R/W	All zeros	24.3.1.2/24-5
0x068	PMC5—Performance monitor counter 5	R/W	All zeros	24.3.2.1/24-9
0x070	PMLCA6—Performance monitor local control register A6	R/W	All zeros	24.3.2.1/24-9
0x074	PMLCB6—Performance monitor local control register B6	R/W	All zeros	24.3.1.2/24-5
0x078	PMC6—Performance monitor counter 6	R/W	All zeros	24.3.2.1/24-9
0x080	PMLCA7—Performance monitor local control register A7	R/W	All zeros	24.3.1.2/24-5
0x084	PMLCB7—Performance monitor local control register B7	R/W	All zeros	24.3.1.2/24-5
0x088	PMC7—Performance monitor counter 7	R/W	All zeros	24.3.2.1/24-9
0x090	PMLCA8—Performance monitor local control register A8	R/W	All zeros	24.3.1.2/24-5
0x094	PMLCB8—Performance monitor local control register B8	R/W	All zeros	24.3.1.2/24-5
0x098	PMC8—Performance monitor counter 8	R/W	All zeros	24.3.2.1/24-9
0x0A0	PMLCA9—Performance monitor local control register A9	R/W	All zeros	24.3.1.2/24-5
0x0A4	PMLCB9—Performance monitor local control register B9	R/W	All zeros	24.3.1.2/24-5
0x0A8	PMC9—Performance monitor counter 9	R/W	All zeros	24.3.2.1/24-9

In addition to these registers, the interrupt control provides four pairs of mask registers that can be used to monitor message, interprocessor, timer, and external interrupts. See Section 11.3.4, “Performance Monitor Mask Registers (PMMRs),” for more information.

24.3.1 Control Registers

This section describes the performance monitor control registers in detail.

24.3.1.1 Performance Monitor Global Control Register (PMGC0)

The performance monitor global control register (PMGC0), shown in Figure 24-2, is a 32-bit register used to control all PMCs.

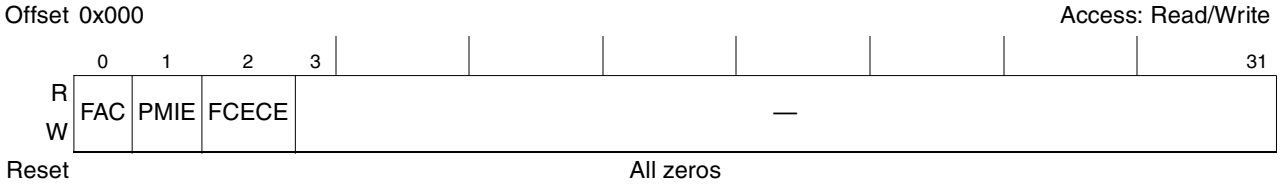


Figure 24-2. Performance Monitor Global Control Register (PMGC0)

Table 24-2 describes PMGC0 fields.

Table 24-2. PMGC0 Field Descriptions

Bits	Name	Description
0	FAC	Freeze all counters. 0 PMCs are incremented (if permitted by other PMGC0/PMLC bits). 1 PMCs are not incremented. Set by hardware when an interrupt is signalled and FCECE =1.
1	PMIE	Performance monitor interrupt enable. Interrupts are caused by PMC overflows. 0 Interrupts are disabled. 1 Interrupts are enabled and occur when an enabled condition or event occurs.
2	FCECE (DISCOUNT)	Freeze counters on enabled condition or event. An enabled condition or event is defined as: The msb = 1 in PMC*n* and PMLCA*n*[CE] = 1. The use of the trigger and freeze counter conditions depends on the enabled condition. 0 PMCs can be incremented (if permitted by other control bits). 1 PMCs can be incremented (if permitted by other control bits) only until an enabled condition or event occurs, at which time PMGC0[FAC] is set. It is up to software to clear FAC.
3–31	—	Reserved

24.3.1.2 Performance Monitor Local Control Registers (PMLCA*n*, PMLCB*n*)

The performance monitor local control registers (PMLCA*n* and PMLCB*n*) are used to control the operation of the PMCs. The performance monitor local control A and B registers are paired 32-bit control registers that are associated with an individual counter to specify how the counter is used and what event is monitored on that counter. Figure 24-3 shows the performance monitor local control A0 register (PMLCA0).

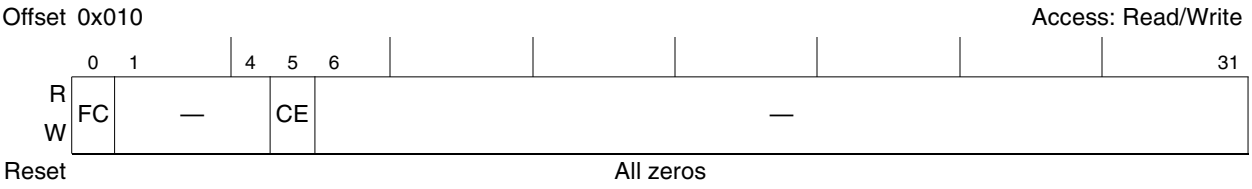


Figure 24-3. Performance Monitor Local Control Register A0 (PMLCA0)

Table 24-3 describes PMLCA0 fields.

Table 24-3. PMLCA0 Field Descriptions

Bits	Name	Description
0	FC	Freeze counter. Basic counter enable. 0 The PMCs are enabled and incremented (if permitted by other SPM control bits). 1 The PMCs are disabled–they do not increment.
1–4	—	Reserved
5	CE	Condition enable. Controls counter overflow condition. Should be cleared when PMC0 is used as a trigger or is selected for chaining. 0 Overflow conditions for PMC0 cannot occur (PMC0 cannot cause interrupts or freeze counters) 1 Overflow conditions occur when PMC0[msb] is set.
6–31	—	Reserved

Figure 24-4 shows the performance monitor local control registers A1–A8.

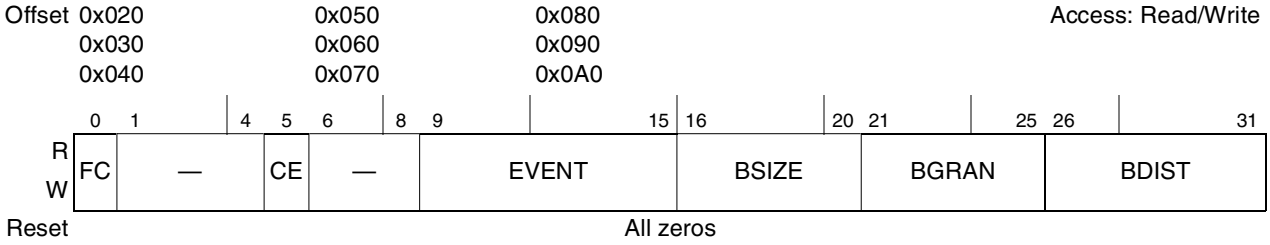


Figure 24-4. Performance Monitor Local Control A Registers (PMLCA1–PMLCA9)

Table 24-4 describes PMLCA*n* fields.

Table 24-4. PMLCA1–PMLCA9 Field Descriptions

Bits	Name	Description
0	FC	Freeze counter 0 The PMCs are incremented (if permitted by other PMC control bits). 1 The PMCs are not incremented (if permitted by other PMC control bits).
1–4	—	Reserved
5	CE	Condition enable 0 Overflow conditions for PMC*n* cannot occur (PMC*n* cannot cause interrupts or freeze counters). Should be cleared when PMC*n* is used as a trigger or is selected for chaining. 1 Overflow conditions occur when PMC*n*[msb] is set.
6–8	—	Reserved
9–15	EVENT	Event selector. Up to 128 events selectable. See Section 24.4.7, “Performance Monitor Events,” for definition of events.
16–20	BSIZE	Burst size. Fewest event occurrences that constitute a burst, that is, a rapid sequence of events followed by a relatively long pause. A value less than two implies regular event counting. Any non-threshold, regular event may be counted in a bursty fashion. See Section 24.4.6, “Burstiness Counting,” for more information.

Table 24-4. PMLCA1–PMLCA9 Field Descriptions (continued)

Bits	Name	Description
21–25	BGRAN	Burst granularity. The maximum number of clock cycles between events that are considered part of a single burst. See Section 24.4.6, “Burstiness Counting.”
26–31	BDIST	Burst distance (used with TBMULT). The number of clock cycles between bursts. Must be set to a value greater than BSIZE for proper burstiness counting behavior. 00_0000 Regular counting

Figure 24-5 shows the performance monitor local control B0 register (PMLCB0).

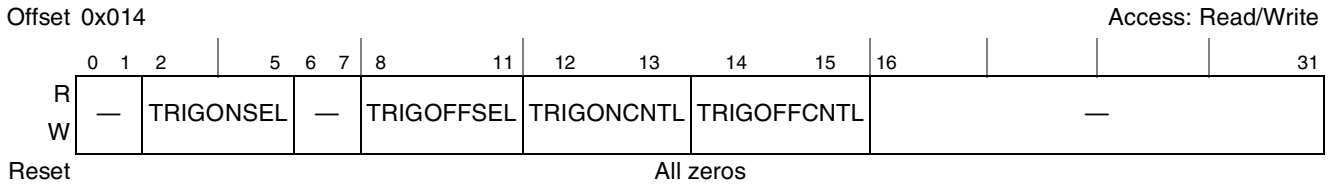


Figure 24-5. Performance Monitor Local Control Register B0 (PMLCB0)

Table 24-5 describes PMLCB0 fields.

Table 24-5. PMLCB0 Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–5	TRIGONSEL	Trigger-on select. The number of the counter that starts event counting. When the specified counter’s TRIGONCNTL event overflows, the current counter begins counting. No triggering occurs if the value is self-referential, that is, when set to the current counter number.
6–7	—	Reserved
8–11	TRIGOFFSEL	Trigger-off select. The number of the counter that stops event counting. When the specified counter’s TRIGONCNTL event overflows, the current counter stops counting. No triggering occurs if the value is self-referential, that is, when set to the current counter number.
12–13	TRIGONCNTL	Trigger-on control. Indicates the condition under which triggering to start counting occurs. 00 Trigger off (no triggering to start) 01 Trigger on change 10 Trigger on overflow 11 Reserved
14–15	TRIGOFFCNTL	Trigger-off control. Indicates the condition under which triggering to stop occurs. 00 Trigger off (no triggering to stop) 01 Trigger on change 10 Trigger on overflow 11 Reserved
16–31	—	Reserved

Figure 24-6 shows performance monitor local control registers B1–B9.

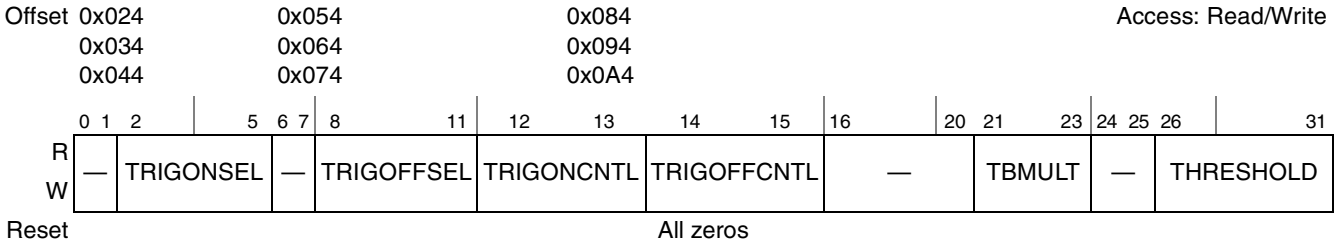


Figure 24-6. Performance Monitor Local Control Register B (PMLCB1–PMLCB9)

Table 24-5 describes PMLCB*n* fields.

Table 24-6. PMLCB*n* Field Descriptions

Bits	Name	Description
0–1	—	Reserved
2–5	TRIGONSEL	Trigger-on select. Set this field equal to the number of the counter that should trigger event counting to start. When the specified counter's TRIGONCNTL event overflows, the current counter begins counting. No triggering occurs when TRIGONSEL = current counter.
6–7	—	Reserved
8–11	TRIGOFFSEL	Trigger-off select. Set this field equal to the number of the counter that should trigger event counting to stop. When the specified counter's TRIGONCNTL event overflows, the current counter stops counting. No triggering occurs when TRIGOFFSEL = current counter.
12–13	TRIGONCNTL	Trigger-on control. Indicates the condition under which triggering to start counting occurs. 00 Trigger off (no triggering to start) 01 Trigger on change 10 Trigger on overflow 11 Reserved
14–15	TRIGOFFCNTL	Trigger-off control. Indicates the condition under which triggering to stop occurs. 00 Trigger off (no triggering to stop) 01 Trigger on change 10 Trigger on overflow 11 Reserved
16–20	—	Reserved
21–23	TBMULT	Threshold and burstiness multiplier. Threshold events are counted when the event duration exceeds a specified threshold value. The threshold is scaled based on the TBMULT settings. The burst distance for burstiness counting is also scaled using the TBMULT settings. For all events that scale the threshold, the threshold field is multiplied by the factors shown below (ranging from 1 to 128). 000 1 001 2 010 4 011 8 100 16 101 32 110 64 111 128

Table 24-6. PMLCB*n* Field Descriptions (continued)

Bits	Name	Description
24–25	—	Reserved
26–31	THRESHOLD	Threshold. Only events whose (number of) occurrences exceed this value are counted. By varying the threshold value, software can characterize the events subject to the threshold. For example, if PMC2 counts BD read latencies for which the duration exceeds the threshold, software can obtain the distribution of BD read latencies for a given program by monitoring the program repeatedly using a different threshold value each time.

24.3.2 Counter Registers

This section describes the PMCs in detail.

NOTE

Because accessing a PMC manually has priority over incrementing it due to event counting, reading or writing a PMC while it is counting may affect the count. Likewise, accessing a performance monitor control register while its target counter is counting may also affect the count.

24.3.2.1 Performance Monitor Counters (PMC0–PMC9)

PMC0–PMC9 are used to count events selected by the performance monitor local control registers. PMC0, shown in Figure 24-7, is associated with two 32-bit registers that form a 64-bit counter designated to count clock cycles. PMC0 upper represents the upper 32 bits of counter 0, and PMC0 lower represents the lower 32 bits.

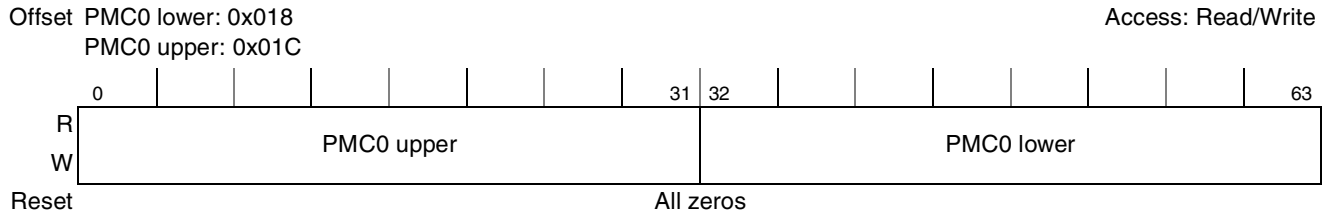


Figure 24-7. Performance Monitor Counter Register 0 (PMC0)

Table 24-7 describes PMC0 fields.

Table 24-7. PMC0 Field Descriptions

Bits	Name	Description
0–63	PMC0	Event count. Counts only clock cycles

PMC1–PMC8, shown in Figure 24-8, are 32-bit counters that can monitor 64 unique events in addition to the 64 reference events that can be counted on all of these registers.

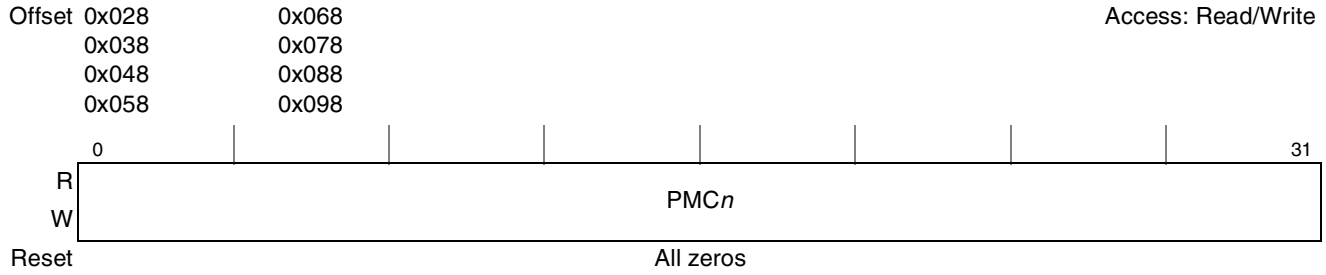


Figure 24-8. Performance Monitor Counter Register (PMC1–PMC8)

Table 24-8 describes PMC*n* fields.

Table 24-8. PMC[1–9] Field Descriptions

Bits	Name	Description
0–31	PMC*n*	Event count. An overflow is indicated when msb = 1. Manually setting the msb can cause an immediate interrupt.

24.4 Functional Description

This section describes the use of some features of the performance monitor.

24.4.1 Performance Monitor Interrupt

PMCs can generate an interrupt on an overflow when the msb of a counter changes from 0 to 1. For the interrupt to be signalled, the condition enable bit (PMLCA*n*[CE]) and performance monitor interrupt enable bit (PMGC0[PMIE]) must be set. When an interrupt is signalled and the freeze-counters-on-enabled-condition-or-event bit (PMGC0[FCECE]) is set, PMGC0[FAC] is set by hardware and all of the registers are frozen. Software can clear the interrupt condition by resetting the performance monitor and clearing the most significant bit of the counter that generated the overflow.

24.4.2 Event Counting

Using the control registers described in Section 24.3.1, “Control Registers,” the nine PMCs can count the occurrences of specific events. The 64-bit PMC0 is designated to count only clock cycles. However, to provide flexibility, a total of 64 reference events can be counted on any of the 32-bit PMCs (PMC1–PMC9). Additionally, up to 64 unique events can be counted on each 32-bit counter.

The performance monitor must be reset before event counting sequences. The performance monitor can be reset by first freezing one or more counters and then clearing the freeze condition to allow the counters to count according to the settings in the performance monitor registers. Counters can be frozen individually by setting PMLCAn[FC] bits, or simultaneously by setting PMGC0[FAC]. Simply clearing these freeze bits will then allow the performance monitor to begin counting based on the register settings.

Note that using PMLCAn[FC] to reset the performance monitor resets only the specified counter. Performance monitor registers can be configured through reads or writes while the counters are frozen as long as freeze bits are not cleared by the register accesses.

24.4.3 Threshold Events

The threshold feature allows characterization of events that can take a variable number of clock cycles to occur. Threshold events are counted only if the latency is greater than the threshold value specified in PMLCB*n*[THRESHOLD].

For duration threshold event sequences, the PMC increments only when the duration of the event is equal to or greater than the threshold value. The threshold value is scaled by a multiple specified in PMLCB*n*[TBMULT].

A threshold event requires two signals: The first indicates when a threshold event sequence begins, and the second indicates when it ends. An internal counter determines when the threshold count is exceeded and when the PMC can increment. This internal counter decrements during a threshold event sequence until it reaches the value of one. A new sequence cannot begin until the current one completes. Additional threshold start signals are ignored during a sequence until a threshold stop signal occurs. If both a start and stop signal are asserted during the same cycle in a current sequence, the stop terminates the current sequence and the start signals the beginning of a new one. However, if both signals are asserted during the same cycle while not in a current event sequence, both signals are ignored. Figure 24-9 is a timing diagram for duration threshold event counting.

An illegal condition exists if the threshold value obtained from PMLCB*n*[THRESHOLD] and PMLCB*n*[TBMULT] is less than two. Under these conditions the intent of threshold counting is ambiguous.

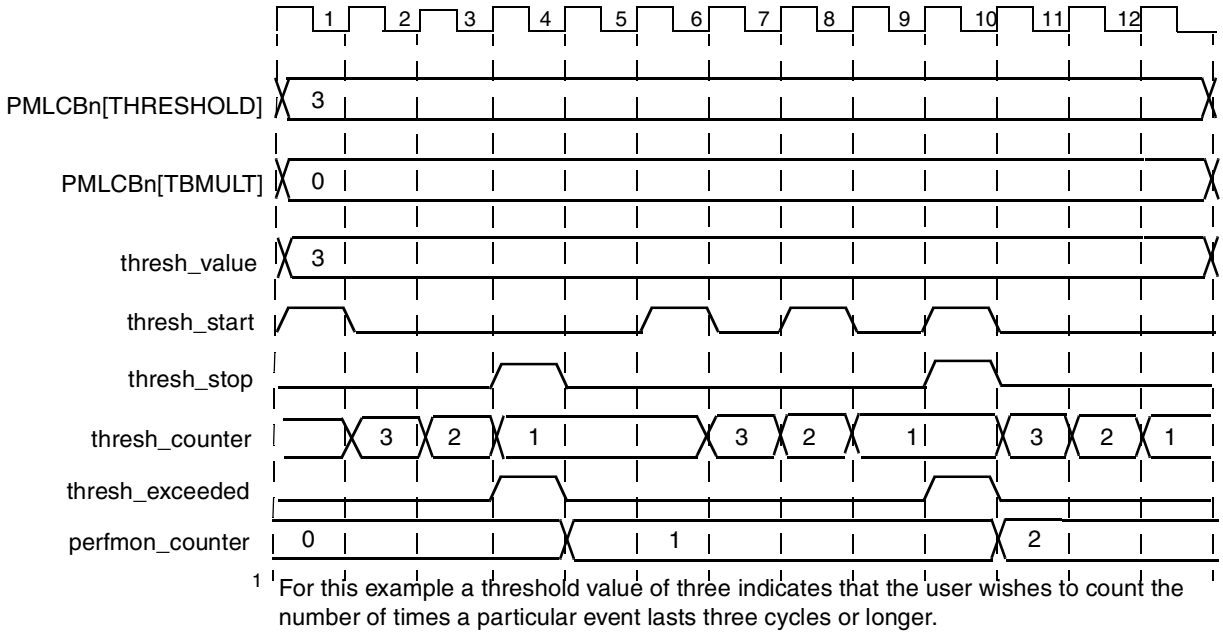


Figure 24-9. Duration Threshold Event Sequence Timing Diagram

The second type of threshold event is the quantity threshold event. For these types of threshold event sequences the performance monitor counter is only incremented when the specified threshold event exceeds the threshold value. These events do not use the multiplier register field (PMLCB*n*[TBMULT]) like the duration threshold events. This type of threshold event is generally used to monitor the usage of buffers and queues. For example, the usage of a specific queue could be characterized by measuring the amount of time the queue is completely full or partially full. For this example the threshold field would be used to specify how many entries are required to be valid in the queue for that event to be counted.

24.4.4 Chaining

By configuring one counter to increment each time another counter overflows, several counters can be chained together to provide event counts larger than 32 bits. Each counter in a chain adds 32 bits to the maximum count. The register chaining sequence is not arbitrary and is specified indirectly by selecting the register overflow event to be counted. Selecting an event has the effect of selecting a source register because all available chaining events, as shown in Table 24-10, are dedicated to specific registers.

Note that the chaining overflow event occurs when the counter reaches its maximum value and wraps, not when the register's msb is set. For this overflow to occur, PMLCA*n*[CE] should be cleared to avoid signalling an interrupt when the counter's most significant bit is set. Note that several cycles may be required for the chained counters to reflect the true count because of the internal delay between when an overflow occurs and a counter increments.

24.4.5 Triggering

Triggering allows one counter to start or stop counting on the change of another counter or on the overflow of another counter. More specifically, if PMC1 is set to start or stop counting as a result of a change or overflow in counter PMC2, then counter PMC2 must be identified in the local control register of counter PMC1. This is done by appropriately setting the trigger-on select bit or trigger-off select bit (PMLCB1[TRIGOFFSEL] or PMLCB1[TRIGONSEL]). Additionally, the condition that triggers the counter must be selected by configuring the corresponding control bits (PMLCB1[TRIGONCNTL] or PMLCB1[TRIGOFFCNTL]). Assuming the counter is enabled by other control register settings, the counter increments (or freezes) when its specified event occurs after the trigger-on (or off) condition occurs.

When trigger on and trigger off are both selected, the trigger-off condition is ignored until the trigger-on condition has occurred. Furthermore, when a trigger-off condition occurs, the counter state is preserved; it is not restarted by subsequent trigger-on conditions.

Triggering is disabled when the counter's trigger-select bits specify itself as the trigger source. Similarly, triggering is disabled when the trigger control bits are cleared.

24.4.6 Burstiness Counting

The burstiness counting feature makes it easier to characterize events that occur in rapid succession followed by a relatively long pause. As shown in Table 24-9, event bursts are defined by size, granularity, and distance.

Table 24-9. Burst Definition

Parameter	Description	Register Field
Size	The minimum number of events constituting a burst	PMLCA*n* [BSIZE]
Granularity	The maximum time between individual events counted as members of the same burst	PMLCA*n* [BGRAN]
Distance	The minimum time between bursts	PMLCA*n* [BDIST] x PMLCB*n*[TBMULT]

Figure 24-10 shows the relationships between size, granularity, and distance. Burstiness counting can be performed for all events except threshold events.

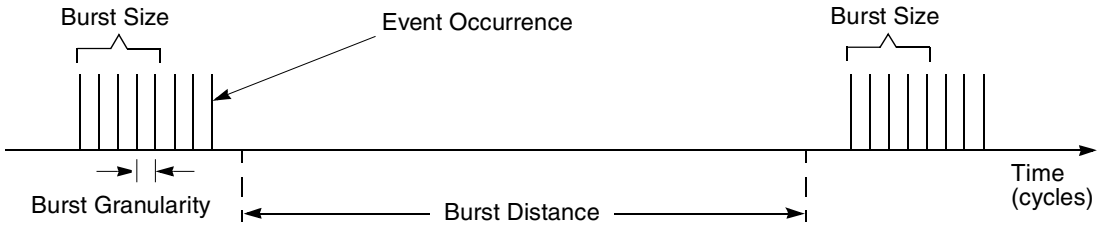


Figure 24-10. Burst Size, Distance, Granularity, and Burstiness Counting

The burstiness size field (PMLCA*n*[BSIZE]) specifies the minimum number of event occurrences that constitute a burst. A burst is identified when the number of event occurrences equals or exceeds PMLCA*n*[BSIZE]. Furthermore, these individual event occurrences must be separated by no more clock cycles than the value in the burstiness granularity field (PMLCA*n*[BGRAN]). Note that, although a burst is identified when the minimum number of events occurs, it is not counted until the burst sequence has ended. A burst sequence ends when the specified burstiness granularity is exceeded, at which point the last valid event has occurred for that sequence.

PMLCA*n*[BGRAN] specifies the maximum number of cycles between individual events for them to qualify as members of the same burst sequence.

The burstiness distance field (PMLCA*n*[BDIST]) and threshold/burstiness multiplier field (PMLCB*n*[TBMULT]) specify the acceptable number of cycles between the end of a burst sequence and the beginning of a new sequence for a group of event occurrences to be counted as an individual burst. The product of the burstiness distance field and the threshold/burstiness multiplier field determine the burstiness distance value used to determine when another burst sequence can begin. Note that the burst distance count begins when a new burst sequence ends and the PMC is incremented. No new burst sequence may begin until the burst distance count has reached zero. After the burst distance count reaches zero, it holds the zero value indicating that a new burst sequence can be counted. The burst distance count begins again when a new burst sequence is identified and counted.

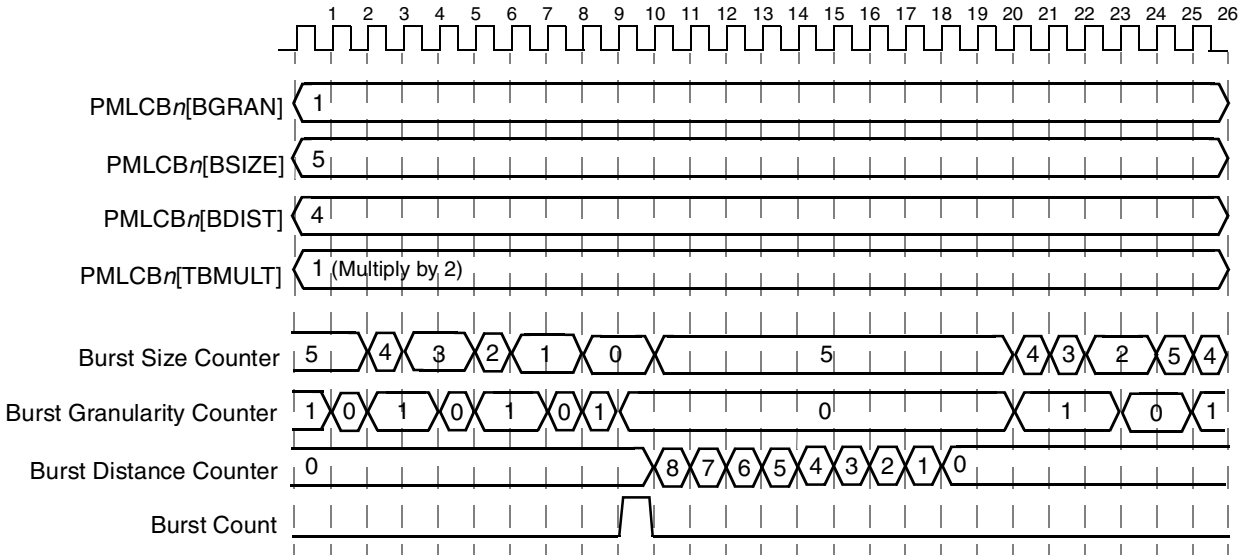
Burstiness counting is disabled when the definition of a burst is ambiguous, that is, when the burst size field is less than two, or the burst distance is zero. When burstiness counting is disabled, regular counting is allowed.

Figure 24-10 shows that the burst distance is measured from the end of one burst sequence and that a new burst sequence may not begin until the burst distance count expires.

Three internal counters track the different values required for burstiness counting.

- Burstiness size is monitored by a counter. It is loaded with the value specified in the local control register when the burst granularity counter and the burst distance counters reach zero, and no new event is occurring. It always decrements when the following conditions occur: its value is not already zero, an event occurs, and the burst distance count equals zero.
- Burstiness granularity is monitored by a counter that is loaded with the specified value in the local control register on the rising edge of an event occurrence whenever the burst distance count equals zero. The granularity counter is decremented (if it has not already reached zero) when an event is not occurring and burst distance count equals zero.
- Burstiness distance is measured by a counter that is loaded with the product of PMLCB*n*[BDIST] and PMLCB*n*[TBMULT] when a burst sequence has been identified and counted. This counter is decremented when burstiness counting is enabled (and the counter has not already reached zero).

A burst is counted at the end of a burst sequence when the three burst parameter counters are all equal to zero. Figure 24-11 shows a burstiness counting example.



1 For this example: count bursts of 5 event occurrences, burst granularity of 1, and acceptable distance between bursts of 8 (product of TBMULT and BDIST).

Figure 24-11. Burstiness Counting Timing Diagram

24.4.7 Performance Monitor Events

This section specifies the performance events that can be counted in PMLCA1–PMLC9. Several tables list these events, as follows:

- Table 24-10, “Event Assignments: No Event, Chaining”
- Table 24-11, “Event Assignments: MCM, DDR Controller, eLBC”
- Table 24-12, “Event Assignments: Interrupt, Debug, Baud Rates”
- Table 24-13, “Event Assignments: DIU”
- Table 24-14, “Event Assignments: Ocean 1, Ocean 2”

In these tables, reference events are shown as R*nn*. All counter-specific events appear only in the column for the counter in which they can be used.

Table 24-10. Event Assignments: No Event, Chaining

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
No Event Selected — PMC Doesn't Count										
Ctr holds current value	R0	R0	R0	R0	R0	R0	R0	R0	R0	Counter input is tied deasserted to prevent counting.
Performance Monitor Chaining Events (PMC Overflow for Cascading of PMC Counters)										
Carry out from PMC0	R1	R1	R1	R1	R1	R1	R1	R1	R1	
Carry out from PMC1	—	R2	R2	R2	R2	R2	R2	R2	R2	
Carry out from PMC2	R3	—	R3	R3	R3	R3	R3	R3	R3	
Carry out from PMC3	R4	R4	—	R4	R4	R4	R4	R4	R4	
Carry out from PMC4	R5	R5	R5	—	R5	R5	R5	R5	R5	
Carry out from PMC5	R6	R6	R6	R6	—	R6	R6	R6	R6	
Carry out from PMC6	R7	R7	R7	R7	R7	—	R7	R7	R7	
Carry out from PMC7	R8	R8	R8	R8	R8	R8	—	R8	R8	
Carry out from PMC8	R9	R9	R9	R9	R9	R9	R9	—	R9	
Carry out from PMC9	R10	R10	R10	R10	R10	R10	R10	R10	—	
Note: A PMC counter is not allowed to receive its own overflow signal.										

Table 24-11. Event Assignments: MCM, DDR Controller, eLBC

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
MCM Events										
MCM Request WaitCore	—	—	—	—	—	—	—	13	—	Asserted for every cycle Core request occurs
MCM GDB Beat	R16	R16	R16	R16	R16	R16	R16	R16	R16	
MCM—"Dispatch from" (Initiator)										
MCM Dispatch	R15	R15	R15	R15	R15	R15	R15	R15	R15	MCM dispatch–includes addr only's from Core. **Note:** All MCM dispatch events are for committed dispatches
MCM Dispatch from Core	16	—	—	—	—	—	—	—	—	MCM dispatch from core –includes addr only's from Core
MCM Dispatch from DIU	—	—	—	—	—	—	—	—	15	MCM dispatch from DIU
MCM Dispatch from PCI	—	—	—	—	—	—	—	—	13	MCM dispatch from PCI

Table 24-11. Event Assignments: MCM, DDR Controller, eLBC (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
MCM Dispatch from PEX4	—	—	—	—	—	—	16	—	—	MCM dispatch from PEX4
MCM Dispatch from PEX8	—	—	—	—	—	—	—	17	—	MCM dispatch from PEX8
MCM Dispatch from DMA1	—	—	—	—	—	—	14	—	—	MCM dispatch from DMA1
MCM Dispatch from DMA2	—	—	—	—	—	—	—	—	18	MCM dispatch from DMA2
MCM Dispatch from Other	—	—	—	—	—	—	—	14	—	MCM dispatch from Other See note on “Other” below.
MCM—Other										
MCM Dispatch Write	17	—	—	—	—	—	—	—	—	MCM dispatch write
MCM Dispatch Read	—	—	—	23	—	—	—	—	—	MCM dispatch read
MCM Dispatch Read Atomic Clr, Set, Dec, Inc	—	—	—	—	—	—	—	—	23	MCM dispatch read clear atomics
MCM Core Read from DDR 1	R17	R17	R17	R17	R17	R17	R17	R17	R17	Core read to DDR port 1
Note: “Other” includes: Boot Seq SAP Core Unused IDs: 001xx 01011 011xx 10100										
DDR Memory Controller Events **Note:** Memory select errors must be enabled (ERR_DISABLE[MSED] = 0) for accurate counting of DDR events.										
Total number of cycles for: Read is returning data from DRAM Write is sending data to DRAM	R11	R11	R11	R11	R11	R11	R11	R11	R11	Asserts once for each beat of data that is transferred from or to the DRAM.
Number of Pipelined *reads* that *missed* in the Row Open Table	57	—	—	—	—	—	—	—	—	Asserts when a *read* that *missed* in the Row Open Table is issued while there is still a previous outstanding *read*.

Table 24-11. Event Assignments: MCM, DDR Controller, eLBC (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Number of Pipelined *reads or writes* that *missed* in the Row Open Table	—	0	—	—	—	—	—	—	—	Asserts when a *read or write* that *missed* in the Row Open Table is issued while there is still a previous outstanding *read or write*.
Number of Non-pipelined *reads* that *missed* in the Row Open Table	—	—	60	—	—	—	—	—	—	Asserts when a *read* that *missed* in the Row Open Table is issued while there are not any outstanding *reads*.
Number of Non-pipelined *reads or writes* that *missed* in the Row Open Table	—	—	—	0	—	—	—	—	—	Asserts when a *read or write* that *missed* in the Row Open Table is issued while there are not any outstanding *reads or writes*.
Number of Pipelined *reads* that *hit* in the Row Open Table	—	—	—	—	56	—	—	—	—	Asserts when a *read* that *hit* in the Row Open Table is issued while there is still a previous outstanding *read*.
Number of Pipelined *reads or writes* that *hit* in the Row Open Table	—	—	—	—	—	0	—	—	—	Asserts when a *read or write* that *hit* in the Row Open Table is issued while there is still a previous outstanding *read or write*.
Number of Non-pipelined *reads* that *hit* in the Row Open Table	—	—	—	—	—	—	57	—	—	Asserts when a *read* that *hit* in the Row Open Table is issued while there are not any outstanding *reads*.
Number of Non-pipelined *reads or writes* that *hit* in the Row Open Table	—	—	—	—	—	—	—	0	—	Asserts when a *read or write* that *hit* in the Row Open Table is issued while there are not any outstanding *reads or writes*.
Total number of Force page closings	—	—	—	1	—	—	—	—	—	
Number of Forced page closings not caused by a refresh	0	—	—	—	—	—	—	—	—	
Total number of Row Open Table *misses*	—	1	—	—	—	—	—	—	—	
Total number of Row Open Table *hits*	—	—	0	—	—	—	—	—	—	
Number of Read-modify-write transactions due to ECC/FCRAM	—	—	—	—	0	—	—	—	—	

Table 24-11. Event Assignments: MCM, DDR Controller, eLBC (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Number of Forced page closings due to collision with bank and sub-bank	R12	R12	R12	R12	R12	R12	R12	R12	R12	
Total number of *reads or writes* from *Core*	R13	R13	R13	R13	R13	R13	R13	R13	R13	
Total number of *reads or writes* from *DIU*	R25	R25	R25	R25	R25	R25	R25	R25	R25	
Total number of *reads or writes* from *PEX4*	—	—	—	3	—	—	—	—	—	
Total number of *reads or writes* from *PEX8*	R22	R22	R22	R22	R22	R22	R22	R22	R22	
Total number of *reads or writes* from *PCI*	—	—	1	—	—	—	—	—	—	
Total number of *reads or writes* from *DMA1*	—	—	—	—	2	—	—	—	—	
Total number of *reads or writes* from *DMA2*	—	—	—	48	—	—	—	—	—	
Total number of *reads* from *Core*	R23	R23	R23	R23	R23	R23	R23	R23	R23	
Number of *Instr reads* from *Core*	40	—	—	—	—	—	—	—	—	
Number of *Data reads* from *Core*	—	—	—	—	—	—	—	—	27	
Total number of *reads* from *DIU*	—	—	—	—	—	52	—	—	—	
Total number of *reads* from *PEX4*	—	—	—	—	—	—	48	—	—	
Total number of *reads* from *PEX8*	—	—	—	—	—	—	—	48	—	
Total number of *reads* from *PCI*	—	—	—	—	—	—	—	—	30	
Total number of *reads* from *DMA1*	42	—	—	—	—	—	—	—	—	
Total number of *reads* from *DMA2*	—	50	—	—	—	—	—	—	—	
Total number of *writes* from *Core*	R24	R24	R24	R24	R24	R24	R24	R24	R24	
Total number of *writes* from *PEX4*	—	—	52	—	—	—	—	—	—	
Total number of *writes* from *PEX8*	—	—	—	—	—	—	39	—	—	

Table 24-11. Event Assignments: MCM, DDR Controller, eLBC (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Total number of *writes* from *PCI*	—	—	—	—	—	—	—	39	—	
Total number of *writes* from *DMA1*	45	—	—	—	—	—	—	—	—	
Total number of *writes* from *DMA2*	—	52	—	—	—	—	—	—	—	
Total number of Row Open Table hits for *reads or writes* from *Core*	R14	R14	R14	R14	R14	R14	R14	R14	R14	
Number of Row Open Table hits for *instr reads* from *Core*	—	51	—	—	—	—	—	—	—	
Number of Row Open Table hits for *data reads or writes* from *Core*	—	—	—	—	—	—	—	—	33	
Total number of Row Open Table hits for *reads or writes* from *DIU*	—	—	—	—	—	1	—	—	—	
Total number of Row Open Table hits for *reads or writes* from *PEX4*	—	—	—	—	—	—	1	—	—	
Total number of Row Open Table hits for *reads or writes* from *PEX8*	—	—	—	—	—	—	—	—	34	
Total number of Row Open Table hits for *reads or writes* from *PCI*	—	—	—	—	—	2	—	—	—	
Total number of Row Open Table hits for *reads or writes* from *DMA1*	—	—	—	—	—	—	—	2	—	
Total number of Row Open Table hits for *reads or writes* from *DMA2*	—	—	54	—	—	—	—	—	—	
Total number of cycles a read is returning data from DRAM	R19	R19	R19	R19	R19	R19	R19	R19	R19	
eLBC Events:										
Accesses hitting Bank 1	51	—	—	—	—	—	—	—	—	Number of accesses hitting the bank selected w/ CS1
Accesses hitting Bank 2	—	56	—	—	—	—	—	—	—	Number of accesses hitting the bank selected w/ CS2
Accesses hitting Bank 3	—	—	55	—	—	—	—	—	—	Number of accesses hitting the bank selected w/ CS3

Table 24-11. Event Assignments: MCM, DDR Controller, eLBC (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Accesses hitting Bank 4	—	—	—	54	—	—	—	—	—	Number of accesses hitting the bank selected w/ CS4
Accesses hitting Bank 5	—	—	—	—	48	—	—	—	—	Number of accesses hitting the bank selected w/ CS5
Accesses hitting Bank 6	—	—	—	—	—	53	—	—	—	Number of accesses hitting the bank selected w/ CS6
Accesses hitting Bank 7	—	—	—	—	—	—	50	—	—	Number of accesses hitting the bank selected w/ CS7
Accesses hitting Bank 8	—	—	—	—	—	—	—	50	—	Number of accesses hitting the bank selected w/ CS8
GPCM Read Cycles	53	—	—	—	—	—	—	—	—	Number of cycles a read is taken in GPCM
UPM Read Cycles	—	58	—	—	—	—	—	—	—	Number of cycles a read is taken in UPM
FCM Read Cycles	—	—	57	—	—	—	—	—	—	Number of cycles - read FCM buffer RAM
GPCM Write Cycles	—	—	—	56	—	—	—	—	—	Number of cycles a write is taken in GPCM
UPM Write Cycles	—	—	—	—	50	—	—	—	—	Number of cycles a write is taken in UPM
FCM Write Cycles	—	—	—	—	—	55	—	—	—	Number of cycles a write is taken in FCM
Correctable FCM ECC Errors	—	—	—	—	—	—	51	—	—	Number of correctable FCM ECC errors
Uncorrectable FCM ECC Errors	—	—	—	—	—	—	—	51	—	Number of uncorrectable FCM ECC errors

Table 24-12. Event Assignments: Interrupt, Debug, Baud Rates

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Interrupt Controller Events										
PIC Total Interrupt Count	R26	R26	R26	R26	R26	R26	R26	R26	R26	Total number of interrupts serviced
PIC Interrupt Active Cycles	—	—	—	—	—	—	—	62	—	Number of cycles there is an active interrupt

Table 24-12. Event Assignments: Interrupt, Debug, Baud Rates (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
PIC Interrupt Service Cycles	—	19	—	—	—	—	—	—	—	Number of cycles there is an interrupt currently being serviced
PIC Interrupt Select 0 (Duration Threshold)	56	—	—	—	—	—	—	—	—	Select 0: Interrupt count over threshold
PIC Interrupt Select 1 (Duration Threshold)	—	—	59	—	—	—	—	—	—	Select 1 Interrupt count over threshold
PIC Interrupt Select 2 (Duration Threshold)	—	—	—	—	55	—	—	—	—	Select 2: Interrupt count over threshold
PIC Interrupt Select 3 (Duration Threshold)	—	—	—	—	—	60	—	—	—	Select 3: Interrupt count over threshold
Debug Events										
External Event	—	—	61	—	—	—	—	—	—	Number of cycles trig_in pin is asserted
Watchpoint Monitor Hits	—	61	—	—	—	—	—	—	—	Number of watchpoint monitor hits
Trace Buffer Hits	58	—	—	—	—	—	—	—	—	Number of trace buffer hits
DUART Events:										
DUART1's UART0 Baud Rate	63	—	—	—	—	—	—	—	—	Real baud rate
DUART1's UART1 Baud Rate	—	—	—	—	63	—	—	—	—	Real baud rate
DUART2's UART0 Baud Rate	—	—	—	—	—	—	—	20	—	Real baud rate
DUART2's UART1 Baud Rate	—	—	—	—	—	—	—	—	11	Real baud rate
IR Events										
IR1's Baud Rate	—	—	58	—	—	—	—	—	—	Real baud rate
SIR (Serial InfraRed) Events										
IR1's SIR Baud Rate	—	—	2	—	—	—	—	—	—	Captures BRM clock which runs at 16 * baud rate. Useful since SIR can auto-detect baud rate from incoming data transmission.
IR2's SIR Baud Rate	—	—	—	—	—	—	27	—	—	Captures BRM clock which runs at 16 * baud rate. Useful since SIR can auto-detect baud rate from incoming data transmission.

Table 24-13. Event Assignments: DIU

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
DIU Events										
Horizontal Sync	—	—	—	—	—	—	45	—	—	—
Vertical Sync	—	—	—	—	—	—	—	45	—	—
Write Output Pixel Buffer	—	48	—	—	—	—	—	—	—	—
Read Output Pixel Buffer	—	—	48	—	—	—	—	—	—	—
Buffer Depth <= 1x256	R27	R27	R27	R27	R27	R27	R27	R27	R27	—
Buffer Depth <= 2x256	—	—	—	49	—	—	—	—	—	—
Buffer Depth <= 3x256	—	—	—	—	45	—	—	—	—	—
Buffer Depth <= 4x256	—	—	—	—	—	40	—	—	—	—
Buffer Depth <= 5x256	—	—	—	—	—	—	—	—	31	—
Buffer Depth <= 6x256	43	—	—	—	—	—	—	—	—	—
Buffer Empty: Plane 1	—	—	53	—	—	—	—	—	—	—
Buffer Empty: Plane 2	—	—	—	53	—	—	—	—	—	—
Buffer Empty: Plane 3	—	—	—	—	38	—	—	—	—	—
Buffer Empty: Write Plane	—	—	—	—	—	42	—	—	—	—
Buffer Full: Plane 1	—	—	—	—	—	48	—	—	—	—
Buffer Full: Plane 2	—	—	—	—	—	—	40	—	—	—
Buffer Full: Plane 3	—	—	—	—	—	—	—	40	—	—
Plane 1 DMA Channel Enabled	46	—	—	—	—	—	—	—	—	—
Plane 2 DMA Channel Enabled	—	53	—	—	—	—	—	—	—	—
Plane 3 DMA Channel Enabled	—	—	—	—	44	—	—	—	—	—
Write Plane DMA Channel Enabled	—	—	—	—	—	47	—	—	—	—
Buffer Descriptor / Lookup Table DMA Channel Enabled	—	—	—	—	—	—	44	—	—	—
Plane 1 DMA Channel Done	—	—	—	—	—	—	—	44	—	—
Plane 2 DMA Channel Done	—	—	—	—	—	—	—	—	26	—
Plane 3 DMA Channel Done	39	—	—	—	—	—	—	—	—	—

Table 24-13. Event Assignments: DIU (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Write Plane DMA Channel Done	—	47	—	—	—	—	—	—	—	—
Buffer Descriptor / Lookup Table DMA Channel Done	—	—	47	—	—	—	—	—	—	—

Table 24-14. Event Assignments: Ocean 1, Ocean 2

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
DMA1 Events										
Channel 0 Read Request	2	—	—	—	—	—	—	—	—	DMA1 channel 0 read request active in system
Channel 1 Read Request	—	5	—	—	—	—	—	—	—	DMA1 channel 1 read request active in system
Channel 2 Read Request	—	—	4	—	—	—	—	—	—	DMA1 channel 2 read request active in system
Channel 3 Read Request	—	—	—	6	—	—	—	—	—	DMA1 channel 3 read request active in system
Channel 0 Write Request	3	—	—	—	—	—	—	—	—	DMA1 channel 0 write request active in system
Channel 1 Write Request	—	6	—	—	—	—	—	—	—	DMA1 channel 1 write request active in system
Channel 2 Write Request	—	—	5	—	—	—	—	—	—	DMA1 channel 2 write request active in system
Channel 3 Write Request	—	—	—	7	—	—	—	—	—	DMA1 channel 3 write request active in system
Channel 0 Descriptor Request	—	—	—	—	41	—	—	—	—	DMA1 channel 0 descriptor request active in system
Channel 1 Descriptor Request	—	—	—	—	—	44	—	—	—	DMA1 channel 1 descriptor request active in system
Channel 2 Descriptor Request	—	—	—	—	—	—	41	—	—	DMA1 channel 2 descriptor request active in system
Channel 3 Descriptor Request	—	—	—	—	—	—	—	41	—	DMA1 channel 3 descriptor request active in system
Channel 0 Read DW or Less	4	—	—	—	53	—	—	—	—	DMA1 channel 0 read doubleword valid

Table 24-14. Event Assignments: Ocean 1, Ocean 2 (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Channel 1 Read DW or Less	—	7	—	—	—	58	—	—	—	DMA1 channel 1 read doubleword valid
Channel 2 Read DW or Less	—	—	6	—	—	—	54	—	—	DMA1 channel 2 read doubleword valid
Channel 3 Read DW or Less	—	—	—	8	—	—	—	52	—	DMA1 channel 3 read doubleword valid
Channel 0 Write DW or Less	5	—	—	—	—	—	—	—	—	DMA1 channel 0 write doubleword valid
Channel 1 Write DW or Less	—	8	—	—	—	—	—	—	—	DMA1 channel 1 write doubleword valid
Channel 2 Write DW or Less	—	—	7	—	—	—	—	—	—	DMA1 channel 2 write doubleword valid
Channel 3 Write DW or Less	—	—	—	9	—	—	—	—	—	DMA1 channel 3 write doubleword valid
DMA2 Events										
Channel 0 Read Request	—	—	—	—	—	25	—	—	—	DMA2 channel 0 read request active in system
Channel 1 Read Request	—	—	—	—	—	—	23	—	—	DMA2 channel 1 read request active in system
Channel 2 Read Request	—	—	—	—	—	—	—	36	—	DMA2 channel 2 read request active in system
Channel 3 Read Request	—	—	30	—	—	—	—	—	—	DMA2 channel 3 read request active in system
Channel 0 Write Request	—	—	—	32	—	—	—	—	—	DMA2 channel 0 write request active in system
Channel 1 Write Request	—	—	—	—	—	—	—	—	4	DMA2 channel 1 write request active in system
Channel 2 Write Request	48	—	—	—	—	—	—	—	—	DMA2 channel 2 write request active in system
Channel 3 Write Request	—	—	21	—	—	—	—	—	—	DMA2 channel 3 write request active in system
Channel 0 Descriptor Request	—	—	—	33	—	—	—	—	—	DMA2 channel 0 descriptor request active in system
Channel 1 Descriptor Request	—	—	—	—	—	—	24	—	—	DMA2 channel 1 descriptor request active in system
Channel 2 Descriptor Request	—	—	—	—	—	—	—	37	—	DMA2 channel 2 descriptor request active in system
Channel 3 Descriptor Request	—	—	—	—	58	—	—	—	—	DMA2 channel 3 descriptor request active in system

Table 24-14. Event Assignments: Ocean 1, Ocean 2 (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Channel 0 Read DW or Less	47	—	—	—	—	—	—	—	—	DMA2 channel 0 read doubleword valid
Channel 1 Read DW or Less	—	—	29	—	—	—	—	—	—	DMA2 channel 1 read doubleword valid
Channel 2 Read DW or Less	—	—	—	31	—	—	—	—	—	DMA2 channel 2 read doubleword valid
Channel 3 Read DW or Less	—	—	—	—	57	—	—	—	—	DMA2 channel 3 read doubleword valid
Channel 0 Write DW or Less	—	—	—	—	—	26	—	—	—	DMA2 channel 0 write doubleword valid
Channel 1 Write DW or Less	—	—	—	—	—	—	7	—	—	DMA2 channel 1 write doubleword valid
Channel 2 Write DW or Less	—	—	—	—	—	—	—	35	—	DMA2 channel 2 write doubleword valid
Channel 3 Write DW or Less	—	—	—	—	—	—	—	—	6	DMA2 channel 3 write doubleword valid
PCI Events **Note:** PCI performance monitor event counting is only accurate while the PCI controller is configured for synchronous operation.										
PCI clock cycles	R28	R28	R28	R28	R28	R28	R28	R28	R28	—
PCI Inbound Memory Reads	62	—	—	—	—	—	—	—	—	Includes all read types.
PCI Inbound Memory Writes	—	37	—	—	—	—	—	—	—	—
PCI Inbound Config Reads	—	—	63	—	—	—	—	—	—	—
PCI Inbound Config Writes	—	—	—	37	—	—	—	—	—	—
PCI Outbound Memory Reads	—	—	—	—	30	—	—	—	—	Includes all read types.
PCI Outbound Memory Writes	—	—	—	—	—	32	—	—	—	—
PCI Outbound I/O Reads	—	—	37	—	—	—	—	—	—	—
PCI Outbound I/O Writes	—	—	—	38	—	—	—	—	—	—
PCI Outbound Config Reads	—	—	—	—	—	—	26	—	—	—
PCI Outbound Config Writes	—	—	—	—	—	—	—	26	—	—

Table 24-14. Event Assignments: Ocean 1, Ocean 2 (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
PCI Inbound 32-bit Read Data Beats	30	—	—	—	—	—	—	—	—	—
PCI Inbound 32-bit Write Data Beats	—	38	—	—	—	—	—	—	—	—
PCI Outbound 32-bit Read Data Beats	—	—	38	—	—	—	—	—	—	—
PCI Outbound 32-bit Write Data Beats	—	—	—	39	—	—	—	—	—	—
PCI Total Transactions	—	—	—	—	—	—	29	—	—	Includes all 32 and 64-bit transactions
PCI Inbound Purgeable Reads	—	2	—	—	—	—	—	—	—	—
PCI Inbound (Speculative Reads) Purgeable Reads Discarded	—	—	—	—	—	—	—	63	—	—
PCI Idle Cycles	31	—	—	—	—	—	—	—	—	—
PCI Dual Address Cycles	—	40	—	—	—	—	—	—	—	—
PCI Internal Cycles	—	—	39	—	—	—	—	—	—	—
PCI Inbound Memory Read	34	—	—	—	—	—	—	—	—	—
PCI Inbound Memory Readline	—	44	—	—	—	—	—	—	—	—
PCI Inbound Memory Read Multiple	—	—	42	—	—	—	—	—	—	—
PCI Outbound Memory Reads	—	—	—	43	—	—	—	—	—	—
PCI Outbound Memory Read Lines	—	—	—	—	36	—	—	—	—	—
PCI Wait	35	—	—	—	—	—	—	—	—	$\overline{\text{PCI\_IRDY}}$, $\overline{\text{PCI\_TRDY}}$ not both asserted
PCI Snoopable	32	—	—	—	—	—	—	—	—	—
PCI Write Stash	—	42	—	—	—	—	—	—	—	—
PCI Write Stash w/ Lock	—	—	41	—	—	—	—	—	—	—
PCI Read Unlock	—	—	—	42	—	—	—	—	—	—
PCI Byte Enable Transactions	33	—	—	—	—	—	—	—	—	—
PCI Non-Byte Enable Transactions	—	43	—	—	—	—	—	—	—	—
n **PCI Express Port 1 Events (4 lane PCI Express):**										
Inbound G2PI read	—	—	—	—	—	—	—	55	—	—
Inbound G2PI write	—	—	—	—	—	—	—	—	37	—

Table 24-14. Event Assignments: Ocean 1, Ocean 2 (continued)

Event	Performance Monitor Counter (PMC) Event Assignments									Description
	PMC 1	PMC 2	PMC 3	PMC 4	PMC 5	PMC 6	PMC 7	PMC 8	PMC 9	
Inbound G2PI data	—	—	—	—	60	—	—	—	—	—
Outbound G2PI read	—	—	—	—	—	62	—	—	—	—
Outbound G2PI write	—	—	—	—	—	—	61	—	—	—
Outbound G2PI data	—	—	—	—	—	—	—	60	—	—
Inbound Static Queue 0 Start (Duration Threshold)	R54	R54	R54	R54	R54	R54	R54	R54	R54	Lifetime of ISQ entry 0 or 6
Outbound Static Queue 0 Start (Duration Threshold)	R55	R55	R55	R55	R55	R55	R55	R55	R55	Lifetime of OSQ entry 0
PCI Express Port 2 Events (8 lane PCI Express): n										
Inbound G2PI read	59	—	—	—	—	—	—	—	—	—
Inbound G2PI write	—	—	—	—	52	—	—	—	—	—
Inbound G2PI data	—	—	—	—	—	57	—	—	—	—
Outbound G2PI read	—	18	—	—	—	—	—	—	—	—
Outbound G2PI write	—	—	—	—	—	—	—	28	—	—
Outbound G2PI data	54	—	—	—	—	—	—	—	—	—
Inbound Static Queue 0 Start (Duration Threshold)	R29	R29	R29	R29	R29	R29	R29	R29	R29	Lifetime of ISQ entry 0 or 6
Outbound Static Queue 0 Start (Duration Threshold)	R30	R30	R30	R30	R30	R30	R30	R30	R30	Lifetime of OSQ entry 0
n **PCI Express Port 1 PIPE Interface Events (4 lane PCI Express):**										
PEX Symbol Disparity Error	—	—	—	—	—	—	—	53	—	—
PEX Symbol Decode Error	—	—	—	—	—	—	—	—	35	—
n **PCI Express Port 2 PIPE Interface Events (8 lane PCI Express):**										
PEX Symbol Disparity Error	14	—	—	—	—	—	—	—	—	—
PEX Symbol Decode Error	—	60	—	—	—	—	—	—	—	—

24.4.8 Performance Monitor Examples

Table 24-16 contains sample register settings for the four supported modes.

- Simple event performance monitoring example
- Triggering event performance monitoring example
- Threshold event performance monitoring example
- Burstiness event performance monitoring example

The settings in Table 24-15 are identical for all four examples.

Table 24-15. PMGC0 and PMLCA*n* Settings

Field	Setting	Reason
PMGC0[FAC]	0	Counters must not be frozen.
PMGC0[PMIE]	1	Performance monitor interrupts are enabled
PMGC0[FCECE]	1	Counters should be frozen when an interrupt is signalled.
PMLCA*n*[FC]	0	Counters cannot be frozen for counting.
PMLCA*n*[CE]	1	Overflow condition enable is required to allow interrupt signalling.

For simple event counting, a non-threshold event is selected in PMLCA*n*[EVENT] and all other features are disabled by clearing all register fields except for CE.

For the triggering example any event can be selected in PMLCA*n*[EVENT]. All other features are disabled by clearing these register fields except for CE to allow interrupt signalling. If PMLCB*n*[TRIGONSEL] is 3 and PMLCB*n*[TRIGOFFSEL] is 5, the counter begins and ends counting based on the conditions in counters three and five. Furthermore, if PMLCB*n*[TRIGONCNTL] is 1, the counter begins counting when PMC3 changes value. According to the setting in PMLCB*n*[TRIGOFFCNTL], the counter ends counting when PMC5 overflows. Also, although the register settings for PMC5 is not shown, PMLCA*n*[CE] for this counter must be cleared so that interrupt signalling is not enabled and the counter does not freeze when it overflows.

For threshold counting, a threshold event must be specified in PMLCA*n*[EVENT]. For this example, the duration threshold value is scaled by two because PMLCB*n*[TBMULT] is one. All other features are disabled by clearing the appropriate fields.

Any non-threshold event can use the burstiness feature. For burstiness counting, values for PMLCA*n*[BSIZE,BGRAN,BDIST] and PMLCB*n*[TBMULT] must be specified.

Table 24-16. Register Settings for Counting Examples

Register	Register Field	Simple Event	Triggering	Threshold	Burstiness
PMGC0	FAC	0	0	0	0
	PMIE	1	1	1	1
	FCECE	1	1	1	1
PMLCA*n*	FC	0	0	0	0
	CE	1	1	1	1
	EVENT	89	68	39	2
	BSIZE	0	0	0	5
	BGRAN	0	0	0	1
	BDIST	0	0	0	8

Table 24-16. Register Settings for Counting Examples (continued)

Register	Register Field	Simple Event	Triggering	Threshold	Burstiness
PMLCB*n*	TRIGONSEL	0	3	0	0
	TRIGOFFSEL	0	5	0	0
	TRIGONCNTL	0	1	0	0
	TRIGOFFCNTL	0	2	0	0
	TBMULT	0	0	0	0
	THRESHOLD	0	0	3	0

The performance monitor must be reset before event counting sequences. The performance monitor can be reset by first freezing one or more counters and then clearing the freeze condition to allow the counters to count according to the settings in the performance monitor registers. Counters can be frozen individually by setting PMLCAn[FC] bits, or simultaneously by setting PMGC0[FAC]. Simply clearing these freeze bits will then allow the performance monitor to begin counting based on the register settings.

Note that using PMLCAn[FC] to reset the performance monitor resets only the specified counter. Performance monitor registers can be configured through reads or writes while the counters are frozen as long as freeze bits are not cleared by the register accesses.

Chapter 25
Debug Features and Watchpoint Facilities

This chapter describes all customer-visible debug modes of the integrated device. The debug features on the device pertain to these interfaces: the PCI Express (PEX) interface, the enhanced local bus controller (eLBC), and the DDR SDRAM interface. In addition to the external interfaces, the device provides triggering capabilities based on user-programmable events. The watchpoint and trace buffer also provide some visibility to internal buses. This chapter also describes context ID registers, useful for software debug, and describes the JTAG access port signals that comply with the IEEE 1149.1 boundary-scan specification.

25.1 Introduction

As shown in the block diagram of Figure 25-1, the device provides the following debug features (listed with references to sections of this chapter that describe them):

- PCI Express interface debug
- DDR SDRAM interface debug (Section 25.4.2, “DDR SDRAM Interface Debug”)
- Enhanced local bus controller (eLBC) debug (Section 25.4.3, “Enhanced Local Bus Interface Debug”)
- Watchpoint monitor and trace buffer debug (Section 25.4.4, “Watchpoint Monitor” and Section 25.4.5, “Trace Buffer”)

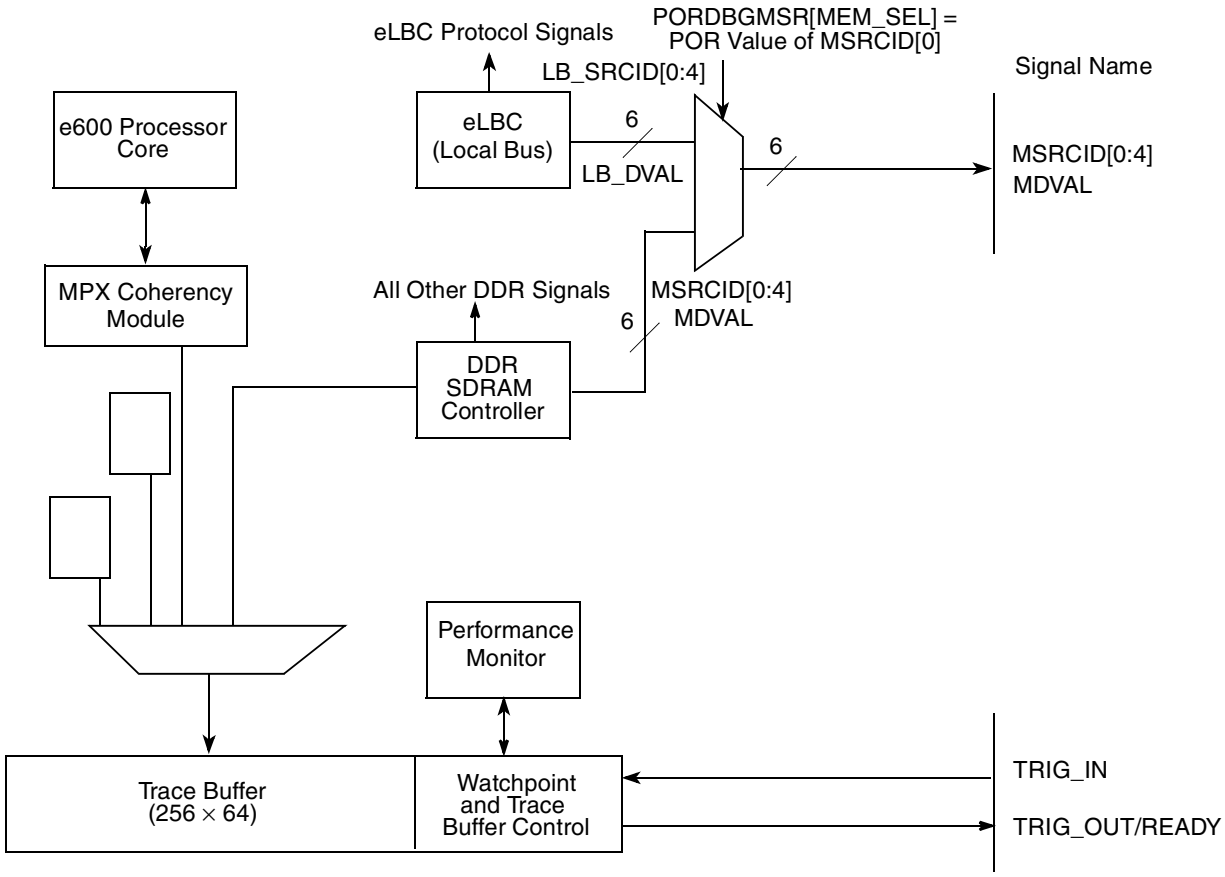


Figure 25-1. Debug and Watchpoint Monitor Block Diagram

25.1.1 Overview

As shown in Figure 25-1, debug information is provided through the following interfaces: PCI, PEX, eLBC, and DDR SDRAM controllers. Limited visibility, through a 256 x 64 trace buffer, is also provided for the processor core interface. This visibility into internal device operation is useful for debugging application software through inverse assembly and reconstruction of the fetch stream.

The combination of a source ID (MSRCID[0:4]) and a data-valid signal (MDVAL) indicates that meaningful debug information is visible on either the enhanced local bus or DDR SDRAM interfaces. A logic analyzer can be programmed to capture data based on the values of MSRCID[0:4] and MDVAL.

Other system debugging is supported by the programmable triggering of the watchpoint monitor and trace buffer. Both can be triggered from one of the following three sources:

- Each other
- A performance monitor event
- An external source (through TRIG_IN).

The watchpoint monitor can be configured to assert TRIG_OUT when a programmed event occurs. The two context ID registers, described in Section 25.3.3, “Context ID Registers,” are useful for software debug.

25.1.2 Features

The principle features of the debug modes and the watchpoint monitor are as follows:

- eLBC and DDR interface source ID and data-valid indicators
  - eLBC or DDR SDRAM source ID can be selected to be driven onto MSRCID[0:4]
- PEX interface: transaction source ID
- Watchpoint monitor that supports
  - Two-level triggering
  - Programmable external trigger (TRIG_OUT)
  - Interlocked with performance monitor to use its large number of counters
- Trace buffer features that supports
  - Two-level triggering
  - Programmable external trigger (TRIG_OUT)
  - Interlocked with performance monitor to use its large number of counters
  - 256-entry trace buffer, 64 bits each
  - Programmable trace start and stop
  - Can function as a second watchpoint monitor
- Context ID registers that can be programmed to trigger events

25.1.3 Modes of Operation

The debug information (source ID and data-valid indicator) on MSRCID[0:4] and MDVAL can be driven with information from the eLBC or from the DDR SDRAM controller. As shown in Table 25-1, the cfg_mem_debug value during POR controls the multiplexing. If cfg_mem_debug is low when sampled during POR, the enhanced local bus information appears on MSRCID[0:4] and MDVAL; otherwise, the DDR SDRAM debug information is presented.

Note that both the watchpoint monitor and trace buffer also operate in a variety of modes.

Table 25-1. POR Configuration Settings and Debug Modes

Configuration Signal	POR Value	Effect	Reference
cfg_mem_debug (MSRCID0)	0	eLBC information appears on MSRCID[0:4] and MDVAL.	
	1	Default value (internal pull-up resistor). DDR information appears on MSRCID[0:4] and MDVAL.	

NOTE

The drivers for the debug signals (MSRCID[0:4] and MDVAL) are controlled by PMUXCR[DBGDRV] in the global utilities block. PMUXCR[DBGDRV] takes its reset value from the cfg_mem_debug configuration input.

This means that if cfg_mem_debug selects eLBC information, PMUXCR[DBGDRV] resets to 0 and the debug signals are driven. This ensures that the debug signals are driven and the information is available while booting from eLBC. After eLBC booting completes, PMUXCR[DBGDRV] can be set to disable the debug signal drivers.

If cfg_mem_debug selects DDR information, DBGDRV resets to 1 and the MSRCID[0:4] and MDVAL are not driven. This ensures that no extra power is consumed by these pins unless explicitly selected by clearing PMUXCR[DBGDRV].

25.1.3.1 Watchpoint Monitor Modes

The watchpoint monitor supports the following operating modes:

- Immediate trigger arming (one-level triggering)—The watchpoint monitor triggers as soon as the first trigger event occurs.
- Wait for trigger arming (two-level triggering)—The watchpoint monitor waits for a specific event before enabling (arming) the trigger logic. The monitor does not respond to trigger events until after the arming event occurs. This function is similar to two-level triggering on a logic analyzer.
- Assert TRIG_OUT on hit—The debug block can be programmed to assert the TRIG_OUT signal when a programmed watchpoint monitor event occurs. This signal can be used to trigger a logic analyzer.

25.1.3.2 Trace Buffer Modes

The trace buffer supports the following operating modes:

- Immediate trigger arming (one-level triggering)—The trace buffer triggers as soon as the first trigger event occurs.
- Wait for trigger arming (two-level triggering)—The trace buffer waits for a specific event before enabling (arming) the trigger logic. The trace buffer does not respond to trigger events until after the arming event occurs. This function is similar to two-level triggering on a logic analyzer.
- Specific interface selection—The trace buffer can be programmed to trace one of several internal interfaces.
- Specific event selection—The trace buffer can be programmed to trace on the occurrence of one or several concurrent events.
- Specific trace selection—To facilitate trace data filtering, the trace buffer can be configured to capture data under the following conditions:
  - On every cycle in which a valid transaction is present on the selected interface

  - Only when a watchpoint monitor event occurs
  - Only when the programmed trace event is detected
- Programmable trace stop—The trace buffer may be programmed to stop tracing when a programmed stop-tracing event occurs or when the 256-entry buffer is full.

25.2 External Signal Description

This section provides information about all the external signals associated with the various debug, JTAG, and test functions. To facilitate system testing, the device provides a JTAG test access port (TAP) that complies with the IEEE 1149.1 boundary-scan specification. This section also describes JTAG TAP signals.

Table 25-2 describes all signals associated with device debug modes.

Table 25-2. Debug Signals—Detailed Signal Descriptions

Signal	I/O	Description	
MDVAL	O	Selectable data-valid. Indicates when valid data is available. May be used by a logic analyzer to capture the data on the data bus.	
		State Meaning	Asserted—Indicates that data is valid on the data bus during the current clock cycle. When the DDR SDRAM interface is selected to source information on MDVAL, this signal is valid for every cycle that data is driven or received on the DDR SDRAM interface. When the eLBC is selected, this signal is valid for every cycle that data is driven or received on the enhanced local bus interface. The assertion of this signal may be used by a logic analyzer to capture data.
		Timing	Asserted/Negated—Referenced to the selected interface, (DDR or enhanced local bus). Asserts when data is valid. Assertions are held for the duration of the transfer. Read data timing is similar to MA. Write data timing is similar to the output MDQ.
MSRCID[0:4]	O	Memory source ID. Attribute signals associated with the memory interface that indicate the source ID for a transaction on an interface. The DDR or enhanced local bus which the debug information applies is specified during POR with MSRCID0 as shown in Table 25-1. One of these signals serves as a reset configuration input signal.	
		State Meaning	Asserted/Negated—In debug mode, always driven with the value of the source ID. The encodings shown in Table 25-29 provide detailed information about a memory transaction.
		Timing	Driven every cycle in debug mode. Similar timing to MA.

Table 25-3 shows detailed descriptions of the watchpoint monitor and trace buffer signals.

Table 25-3. Watchpoint and Trigger Signals—Detailed Signal Descriptions

Signal	I/O	Description	
TRIG_IN	I	Trigger in. Can be used to trigger the watchpoint and trace buffers. Note this is an active-high (rising-edge triggered) signal.	
		State Meaning	Asserted—Indicates that a programmed/armed external event has been detected. Assertion may be used internally to trigger trace buffers and watchpoint mechanisms.
		Timing	Assertion/Negation—The device interprets TRIG_IN as asserted on detection of the rising edge. It may occur at any time. Must remain asserted for at least 3 system clocks to be recognized internally.
TRIG_OUT	O	Trigger out. Function determined by TOSR[SEL]. When TOSR[SEL] is non-zero, it can be used for triggering external devices, like a logic analyzer, with either the watchpoint monitor, the trace buffer, or the performance monitor as trigger sources. When TOSR[SEL] is cleared, TRIG_OUT is multiplexed with READY, which indicates the operational readiness of the device (running or in low-power or debug modes). See Chapter 4, "Reset, Clocking, and Initialization," and Chapter 23, "Global Utilities," for more details about reset, low-power, and debug states.	
		State Meaning	Asserted—When TOSR[SEL] is all zeros, serves as the READY signal, indicating that the device is not in a low-power or debug mode and that it has emerged from reset. SEL ≠ 0 indicates that a programmed trigger event has occurred. Negation—No final watchpoint match condition
		Timing	Assertion may occur at any time. Remains asserted for at least 3 system clocks

Table 25-4 shows detailed descriptions of the JTAG test and other signals.

Table 25-4. JTAG Test and Other Signals—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. 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 standard 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 standard for more details.
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 standard for more details.

Table 25-4. JTAG Test and Other Signals—Detailed Signal Descriptions (continued)

Signal	I/O	Description	
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 standard 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 standard for more details.
$\overline{\text{LSSD\_MODE}}$	I	Used for factory test. Refer to the *MPC8610 Integrated Host Processor Hardware Specifications* for proper treatment.	
TEST_MODFE[0:1]	I	Used for factory test. Refer to the *MPC8610 Integrated Host Processor Hardware Specifications* for proper treatment.	

25.3 Memory Map/Register Definition

The debug and watchpoint monitor registers are accessed by reading and writing to an address comprised of the base address (specified in the CCSRBAR), plus the block base address, plus the offset of the specific register to be accessed. For the debug and watchpoint monitor registers, the block base address is 0xE_2000.

Table 25-5 shows the memory-mapped debug and watchpoint registers of the device. It lists the offset, name, and a cross-reference to the complete description of each register. Undefined 4-byte address spaces within offset 0x000–0xFFF are reserved. Note that all registers must be accessed as aligned 4-byte quantities.

Table 25-5. Debug and Watchpoint Monitor Memory Map

Debug and Watchpoint Monitor—Block Base Address 0xE_2000				
Local Memory Offset	**Register**	**Access**	**Reset**	**Section/Page**
Watchpoint Monitor Registers				
0x000	WMCR0—Watchpoint monitor control register 0	R/W	All zeros	25.3.1.1/25-8
0x004	WMCR1—Watchpoint monitor control register 1	R/W	All zeros	25.3.1.2/25-10
0x008	WMAHR—Watchpoint monitor address high register	R/W	All zeros	25.3.1.3/25-11
0x00C	WMAR—Watchpoint monitor address register	R/W	All zeros	25.3.1.4/25-11

Table 25-5. Debug and Watchpoint Monitor Memory Map (continued)

Debug and Watchpoint Monitor—Block Base Address 0xE_2000				
Local Memory Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x010	WMAMHR—Watchpoint monitor address mask high register	R/W	All zeros	25.3.1.5/25-12
0x014	WMAMR—Watchpoint monitor address mask register	R/W	All zeros	25.3.1.6/25-12
0x018	WMTMR—Watchpoint monitor transaction mask register	R/W	All zeros	25.3.1.7/25-12
0x01C	WMSR—Watchpoint monitor status register	R/W	All zeros	25.3.1.8/25-14
Trace Buffer Registers				
0x040	TBCR0—Trace buffer control register 0	R/W	All zeros	25.3.2.1/25-15
0x044	TBCR1—Trace buffer control register 1	R/W	All zeros	25.3.2.2/25-17
0x048	TBAHR—Trace buffer address high register	R/W	All zeros	25.3.2.3/25-17
0x04C	TBAR—Trace buffer address register	R/W	All zeros	25.3.2.4/25-18
0x050	TBAMHR—Trace buffer address mask high register	R/W	All zeros	25.3.2.5/25-18
0x054	TBAMR—Trace buffer address mask register	R/W	All zeros	25.3.2.6/25-19
0x058	TBTMR—Trace buffer transaction mask register	R/W	All zeros	25.3.2.7/25-19
0x05C	TBSR—Trace buffer status register	R/W	All zeros	25.3.2.8/25-20
0x060	TBACR—Trace buffer access control register	R/W	All zeros	25.3.2.9/25-20
0x064	TBADHR—Trace buffer access data high register	R/W	All zeros	25.3.2.10/25-21
0x068	TBADR—Trace buffer access data register	R/W	All zeros	25.3.2.11/25-21
Context ID Registers				
0x0A0	PCIDR—Programmed context ID register	R/W	All zeros	25.3.3.1/25-22
0x0A4	CCIDR—Current context ID register	R/W	All zeros	25.3.3.2/25-23
Other Registers				
0x0B0	TOSR—Trigger output source register	R/W	All zeros	25.3.4.1/25-23

25.3.1 Watchpoint Monitor Register Descriptions

The following sections describe the control registers for the watchpoint monitor facility.

25.3.1.1 Watchpoint Monitor Control Register 0 (WMCR0)

Watchpoint monitor control register 0 (WMCR0), shown in Figure 25-2, controls the specification of watchpoint monitor events.

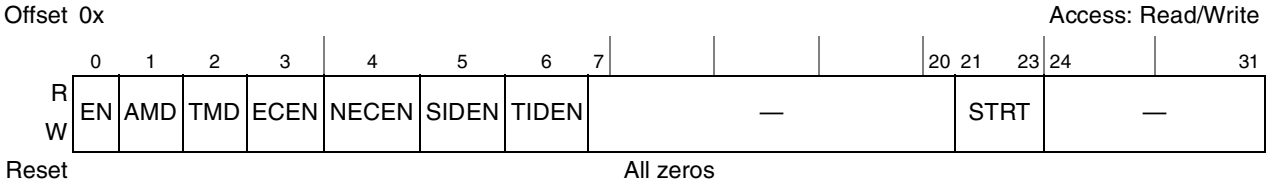


Figure 25-2. Watchpoint Monitor Control Register 0 (WMCR0)

Table 25-6 describes WMCR0 fields.

Table 25-6. WMCR0 Field Descriptions

Bits	Name	Description
0	EN	Watchpoint monitor enable 0 Watchpoint monitor is not enabled; events are not flagged. 1 Watchpoint monitor is enabled; events are flagged.
1	AMD	Address match disable. Qualifies address match as a watchpoint event criterion. 0 Address matching is used to recognize a watchpoint event. 1 Address matching does not affect watchpoint event detection.
2	TMD	Transaction match disable. Qualifies transaction type match (as defined in WMCR1[IFSEL] and WMTMR) as a watchpoint event criterion. 0 A transaction type match is used to recognize watchpoint events. 1 A transaction type match does not affect watchpoint event detection.
3	ECEN	Equal context enable. Qualifies the matching of current context with programmed context as a watchpoint event criterion, as written in the context registers described in Section 25.3.3, “Context ID Registers.” 0 Current context match does not affect watchpoint event detection. 1 Watchpoint events are qualified by comparing current context with the programmed context event value. **Note:** ECEN and NECEN must not be enabled in the same run. If both are set, watchpoint events are inhibited (never occur).
4	NECEN	Not equal context enable. Qualifies the matching of current context with programmed context as a watchpoint event criterion, as written in the context registers described in Section 25.3.3, “Context ID Registers.” 0 The failure of a current context match does not affect watchpoint event detection 1 Watchpoint events are qualified with NOT getting a current context compare with the programmed context event value. **Note:** ECEN and NECEN must not be enabled in the same run. If both are set, watchpoint events are inhibited (never occur).
5	SIDEN	Source ID enable 0 Source ID does not affect watchpoint event detection. 1 Watchpoint events are qualified by comparison with the programmed WMCR1(SID) value.
6	TIDEN	Target ID enable 0 Target ID does not affect watchpoint event detection. 1 Watchpoint events are qualified by comparison with the programmed WMCR1(TID) value.
7–20	—	Reserved

Table 25-6. WMCR0 Field Descriptions (continued)

Bits	Name	Description
21–23	STRT	Start condition. Specifies the event that arms the watchpoint monitor to start looking for the programmed event. 000 No event. Armed immediately 001 Trace buffer event is detected 010 Performance monitor signals overflow 011 TRIG_IN transitions from 0 to 1 100 TRIG_IN transitions from 1 to 0 101 Current context ID equals programmed context ID 110 Current context ID is not equal to programmed context ID 111 Reserved
24–31	—	Reserved

25.3.1.2 Watchpoint Monitor Control Register 1 (WMCR1)

Watchpoint monitor control register 1 (WMCR1), shown in Figure 25-3, controls the specification of watchpoint monitor events.

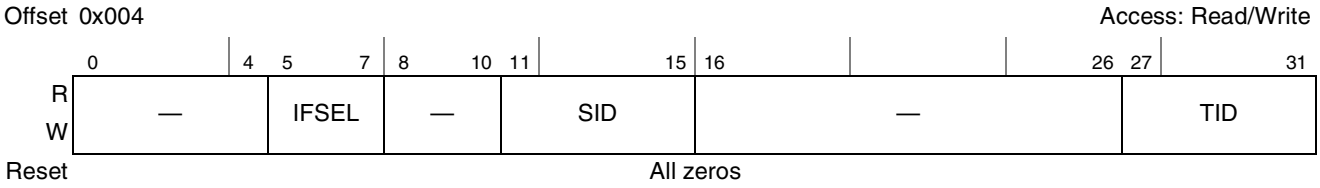


Figure 25-3. Watchpoint Monitor Control Register 1 (WMCR1)

Table 25-7 describes the WMCR1 fields.

Table 25-7. WMCR1 Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–7	IFSEL	Interface selection. Selects the address, transaction type (as defined in WMTMR), and other attributes to be used for comparison 000 Select coherency module (MCM) dispatch interface 001 Select internal DDR SDRAM interface 010 Select PCI interface 011 Select PEX2 (×8) interface 100 Select PEX1 (×4) interface 101–111 Reserved
8–10	—	Reserved
11–15	SID	Source ID. Specifies the source ID associated with WMCR0[SIDEN]. For a definition of the source ID, see Table 25-29.
16–26	—	Reserved
27–31	TID	Target ID. Specifies the target ID associated with WMCR0[TIDEN]. For a definition of the target ID see Table 25-29.

25.3.1.3 Watchpoint Monitor Address High Register (WMAHR)

The watchpoint monitor address high register (WMAHR) shown in Figure 25-4 contains the high-order address bits of the address to match against if WMCR[AMD] is clear.

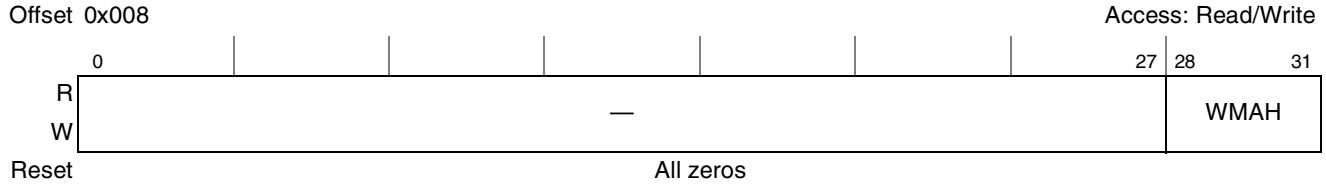


Figure 25-4. Watchpoint Monitor Address High Register (WMAHR)

Table 25-8 describes the WMAHR fields.

Table 25-8. WMAHR Field Descriptions

Bit	Name	Description
0–27	—	Reserved
28–31	WMAH	Watchpoint monitor address high. High-order bits of the match address. A value of 0 masks the address comparison for the corresponding address bit. These correspond to bits 0:3 of the real or platform address.

25.3.1.4 Watchpoint Monitor Address Register (WMAR)

The watchpoint monitor address register (WMAR) shown in Figure 25-5 contains the low-order address bits of the address to match against if WMCR[AMD] is clear. Note that this address may be further qualified with the bits described in Section 25.3.1.6, “Watchpoint Monitor Address Mask Register (WMAMR).”

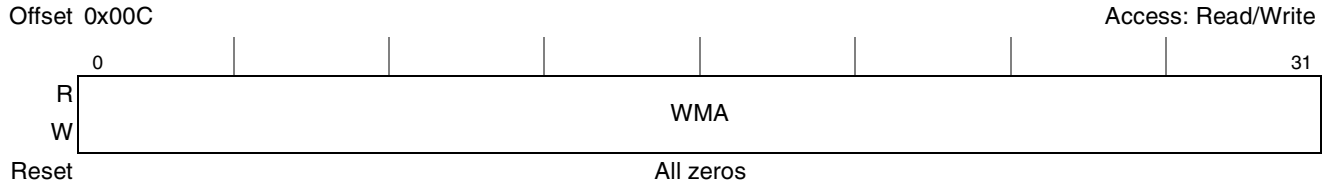


Figure 25-5. Watchpoint Monitor Address Register (WMAR)

Table 25-9 describes the WMAR fields.

Table 25-9. WMAR Field Descriptions

Bits	Name	Description
0–31	WMA	Watchpoint monitor address. Low-order bits of the match address. These correspond to bits 4:35 of the real or platform address.

25.3.1.5 Watchpoint Monitor Address Mask High Register (WMAMHR)

The watchpoint monitor address mask high register (WMAMHR) shown in Figure 25-6 contains the mask for the high-order address bits in the WMAHR. Permits user to mask address bits from the comparison.

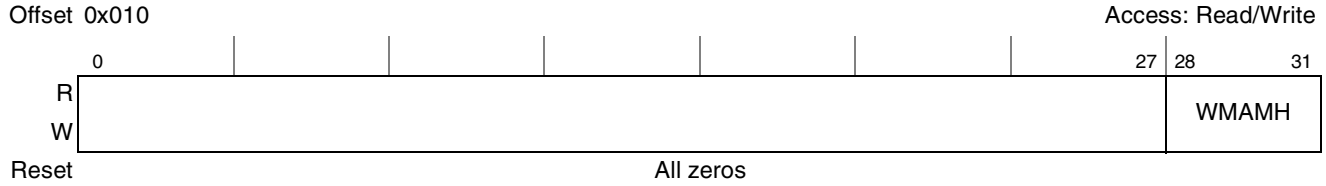


Figure 25-6. Watchpoint Monitor Address Mask High Register (WMAMHR)

Table 25-10 describes the WMAMHR fields.

Table 25-10. WMAMHR Field Descriptions

Bit	Name	Description
0–27	—	Reserved
28–31	WMAMH	Watchpoint monitor address mask high. High-order bits of the match address mask.

25.3.1.6 Watchpoint Monitor Address Mask Register (WMAMR)

The watchpoint monitor address mask register (WMAMR) shown in Figure 25-7 contains the low-order address bits of the mask for the address in the WMAR.

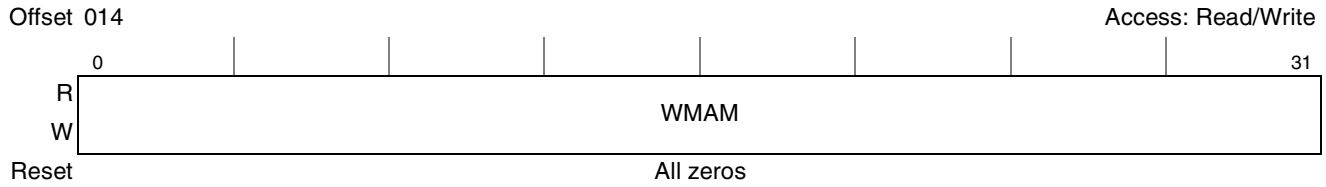


Figure 25-7. Watchpoint Monitor Address Mask Register (WMAMR)

Table 25-11 describes the WMAMR fields.

Table 25-11. WMAMR Field Descriptions

Bits	Name	Description
0–31	WMAM	Watchpoint monitor address mask. Low-order bits of the match address mask. A value of zero masks the address comparison for the corresponding address bit.

25.3.1.7 Watchpoint Monitor Transaction Mask Register (WMTMR)

The watchpoint monitor transaction mask register (WMTMR), shown in Figure 25-8, specifies which transaction types to monitor. WMTMR allows users to qualify watchpoint events specifically with any combination of transaction types. As shown in Table 25-13, each bit represents as many as four separate transaction types; one for each interface. Setting a bit enables watchpoint monitoring for the corresponding transaction types.

Because the supported transaction types vary by interface, the type designated by a WMTMR field also depends on the interface specified by WMCR1[IFSEL]. Table 25-13 lists transaction types associated with each WMTMR bit by interface.

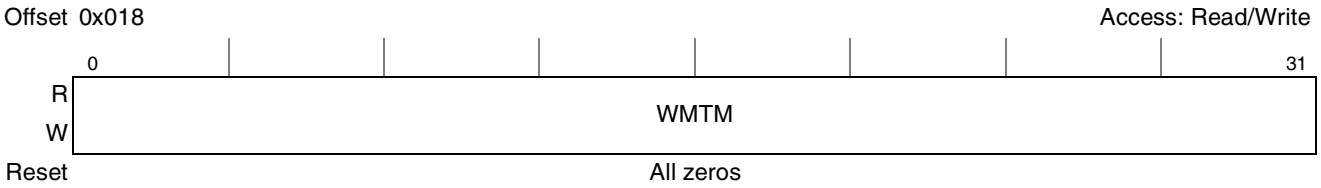


Figure 25-8. Watchpoint Monitor Transaction Mask Register (WMTMR)

Table 25-12 describes the WMTMR fields.

Table 25-12. WMTMR Field Descriptions

Bits	Name	Description
0–31	WMTM	Watchpoint monitor transaction mask. Each bit corresponds to a transaction type as defined in Table 25-13. The transaction associated with any particular bit may be different depending on the interface being monitored. A value of 1 for a given mask bit enables the matching of the transaction associated with that bit. These bits are meaningful only when WMCR0[TMD]=0.

Table 25-13 defines the transactions associated with each transaction mask bit for the different interfaces supported by the watchpoint monitor.

Table 25-13. Transaction Types by Interface

Bit	Description			
	MPX Coherency Module (MCM)	DDR Controller	PCI Express Outbound Transaction	PCI Outbound Transaction
0	Write with local processor snoop	Write	Posted write	Memory write
1	Write with no local processor snoop	Reserved	Non-posted write	I/O write
2	Reserved	Write with allocate	Reserved	
3		Write with allocate and lock		
4–7	Reserved			
8	Read with local processor snoop	Read	Read	Memory read
9	Read with no local processor snoop	Reserved	Reserved	I/O read
10	Reserved	Read with unlock	Reserved	
11–15	Reserved			

Table 25-13. Transaction Types by Interface (continued)

Bit	Description			
	MPX Coherency Module (MCM)	DDR Controller	PCI Express Outbound Transaction	PCI Outbound Transaction
16	ATOMIC clear	ATOMIC clear	Reserved	
17	ATOMIC set	ATOMIC set		
18	ATOMIC decrement	ATOMIC decrement		
19	ATOMIC increment	ATOMIC increment		
20–24	Reserved			
25	Address-only transaction	Reserved		
26–31	Reserved			

25.3.1.8 Watchpoint Monitor Status Register (WMSR)

The watchpoint monitor status register (WMSR) shown in Figure 25-9 indicates the state of the watchpoint monitor.

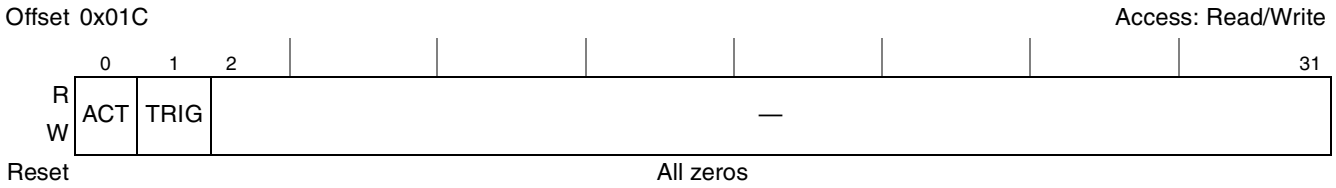


Figure 25-9. Watchpoint Monitor Status Register (WMSR)

Table 25-14 describes the WMSR fields.

Table 25-14. WMSR Field Descriptions

Bits	Name	Description
0	ACT	Active 0 The start triggering event has not occurred; watchpoint monitor is not armed. 1 The start triggering event has occurred; watchpoint monitor is armed.
1	TRIG	Triggered 0 The programmed event in WMCR0 has not yet been triggered. 1 The programmed event in WMCR0 has been triggered at least once.
2–31	—	Reserved

25.3.2 Trace Buffer Register Descriptions

The following sections describes the trace buffer registers.

25.3.2.1 Trace Buffer Control Register 0 (TBCR0)

Trace buffer control register 0 (TBCR0), shown in Figure 25-10, specifies trace buffer events.

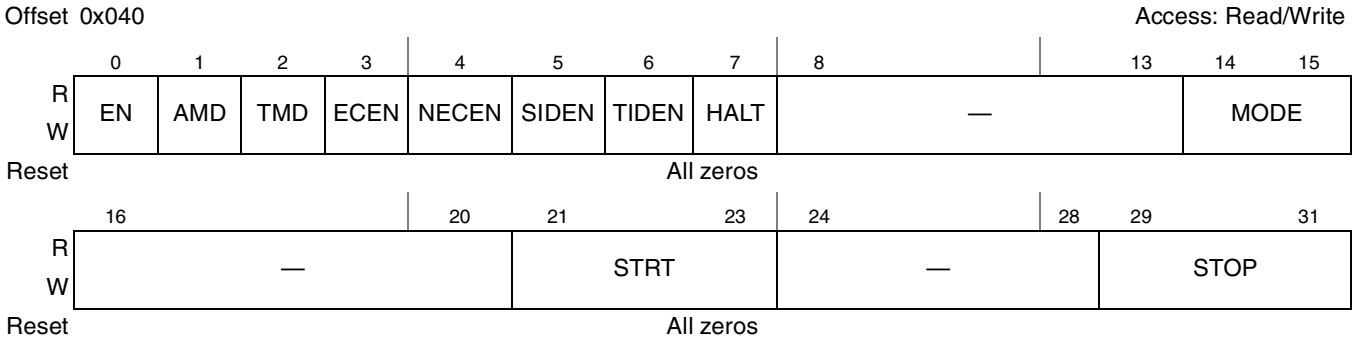


Figure 25-10. Trace Buffer Control Register 0 (TBCR0)

Table 25-15 describes the TBCR0 fields.

Table 25-15. TBCR0 Field Descriptions

Bits	Name	Description
0	EN	Enable 0 The trace buffer facility is disabled. 1 The trace buffer facility is enabled.
1	AMD	Address match disable 0 The address match is used to qualify a trace buffer event. 1 The address match is ignored when detecting a trace buffer event.
2	TMD	Transaction match disable 0 The transaction type match is used to qualify a trace buffer event. 1 The transaction type match is ignored when detecting a trace buffer event.
3	ECEN	Equal context enable. Qualifies the matching of current context with programmed context as a trace buffer event criterion, as written in the context registers described in Section 25.3.3, “Context ID Registers.” 0 Current context match does not affect trace buffer event detection 1 Trace buffer events are qualified by comparing current context with the programmed context event value. **Note:** ECEN and NECEN must not be enabled in the same run. If both are set, trace buffer events are inhibited (never occur).
4	NECEN	Not equal context enable. Qualifies the matching of current context with programmed context as a trace buffer event criterion, as written in the context registers described in Section 25.3.3, “Context ID Registers.” 0 The failure of a current context match does not affect trace buffer event detection 1 trace buffer events are qualified with NOT getting a current context compare with the programmed context event value. **Note:** ECEN and NECEN must not be enabled in the same run. If both are set, trace buffer events are inhibited (never occur).

Table 25-15. TBCR0 Field Descriptions (continued)

Bits	Name	Description
5	SIDEN	Source ID enable 0 Trace buffer events ignore the programmed source ID value. 1 Trace buffer events are qualified by comparison with the programmed SID event value.
6	TIDEN	Target ID enable 0 Trace buffer events ignore the programmed TID event value. 1 Trace buffer events are qualified by comparison with the programmed TID event value. This comparison only applies when the MCM is selected for tracing (TBCR1[IFSEL] is all zeros).
7	HALT	Halt causes the trace buffer to stop tracing immediately.
8–13	—	Reserved
14–15	MODE	Trace mode. Specifies one of two trace modes. 00 Trace every valid transaction 01 Reserved 10 Trace only cycles in which a trace event is detected. Note that if EN and other TBCR0 fields are not properly programmed to specify a traceable event, tracing occurs for every valid address. 11 Reserved
16–20	—	Reserved
21–23	STRT	Start condition. Specifies the event that arms the trace buffer to start looking for the programmed event 000 No event. Armed immediately 001 Watchpoint monitor event is detected 010 Trace buffer event is detected 011 Performance monitor signals overflow 100 TRIG_IN transitions from 0 to 1 101 TRIG_IN transitions from 1 to 0 110 Current context ID equals programmed context ID 111 Current context ID does not equal programed context ID
24–28	—	Reserved
29–31	STOP	Trace stop mode. Specifies the event that stops the updating of the trace buffer after it has been started. Trace buffer only stops after it has been triggered at least once. 000 Buffer is full 001 Watchpoint monitor event is detected 010 Trace buffer event is detected 011 Performance monitor signals overflow 100 TRIG_IN transitions from 0 to 1 101 TRIG_IN transitions from 1 to 0 110 Current context ID equals programmed context ID 111 Current context ID does not equal programed context ID

25.3.2.2 Trace Buffer Control Register 1 (TTBCR1)

Trace buffer control register 1 (TBCR1), shown in Figure 25-11, specifies trace buffer events.

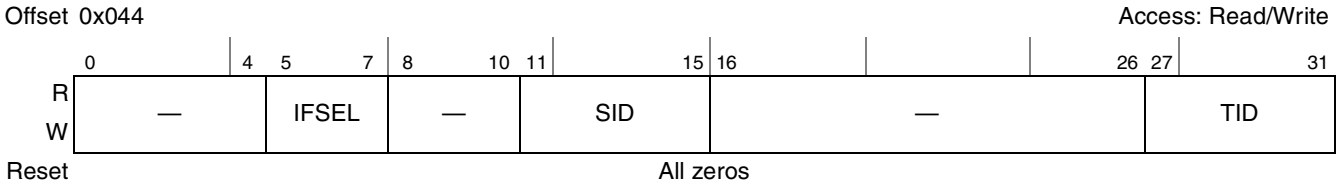


Figure 25-11. Trace Buffer Control Register 1 (TBCR1)

Table 25-16 describes the TBCR1 fields.

Table 25-16. TBCR1 Field Descriptions

Bits	Name	Description
0–4	—	Reserved
5–7	IFSEL	Interface selection. Specifies the interface that sources information for both comparison/buffer control and buffer data capture. 000 Select coherency module (MCM) dispatch interface 001 Select internal DDR SDRAM interface 010 Select PCI interface 011 Select PEX2 (×8) interface 100 Select PEX1 (×4) interface 101–111 Reserved
8–10	—	Reserved
11–15	SID	Source ID. Specifies the source ID associated with TBCR0[SIDEN]. The source ID is defined in Table 25-29.
16–26	—	Reserved
27–31	TID	Target ID. Specifies the target ID associated with TBCR0[TIDEN]. The target ID is defined in Table 25-29.

25.3.2.3 Trace Buffer Address High Register (TBAHR)

The trace buffer address high register (TBAHR) shown in Figure 25-12 contains the high-order address bits of the address to match against if TBCR[AMD] is clear.

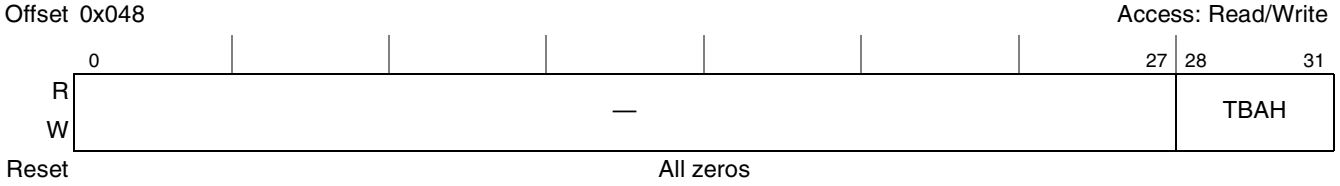


Figure 25-12. Trace Buffer Address High Register (TBAHR)

Table 25-17 describes the TBAHR fields.

Table 25-17. TBAHR Field Descriptions

Bit	Name	Description
0–27	—	Reserved
28–31	TBAH	Trace buffer address high. High-order bits of the match address.

25.3.2.4 Trace Buffer Address Register (TBAR)

The trace buffer address register (TBAR) shown in Figure 25-13 contains the low-order address bits of the address to match against (if TBCR[AMD] is zero). This address may be further qualified by the mask bits defined in Section 25.3.2.6, "Trace Buffer Address Mask Register (TBAMR)."

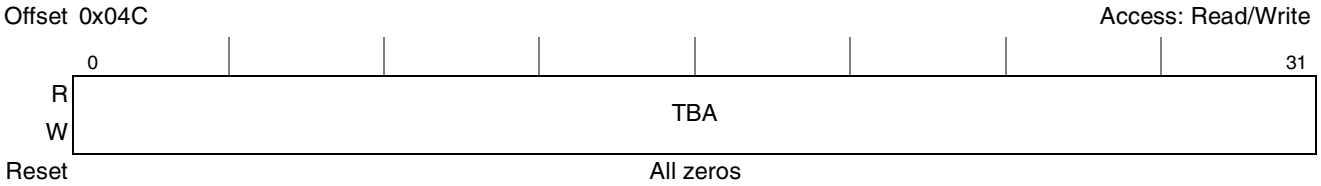


Figure 25-13. Trace Buffer Address Register (TBAR)

Table 25-18 describes the TBAR field.

Table 25-18. TBAR Field Descriptions

Bits	Name	Description
0–31	TBA	Trace buffer address. Low-order bits of the match address.

25.3.2.5 Trace Buffer Address Mask High Register (TBAMHR)

The trace buffer address mask high register (TBAMHR) shown in Figure 25-14 contains the mask for the high-order address bits in the TBAHR. Allows excluding address bits from the comparison.

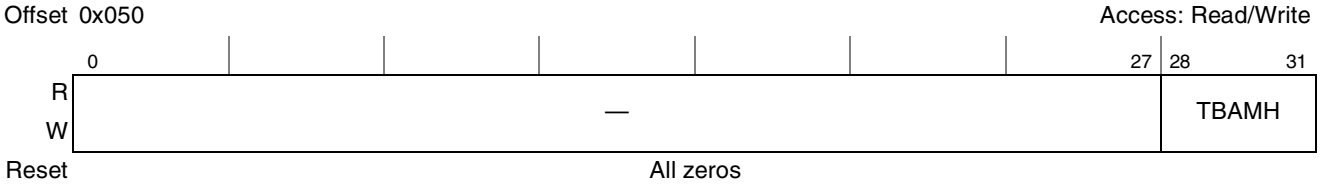


Figure 25-14. Trace Buffer Address High Register (TBAMHR)

Table 25-19 describes the TBAMHR fields.

Table 25-19. TBAMHR Field Descriptions

Bits	Name	Description
0–27	—	Reserved
28–31	TBAMH	Trace buffer address mask high. High-order bits of the match address mask.

25.3.2.6 Trace Buffer Address Mask Register (TBAMR)

The trace buffer address mask register (TBAMR) shown in Figure 25-15 contains a mask for the TBAR, which allows excluding address bits from the comparison.

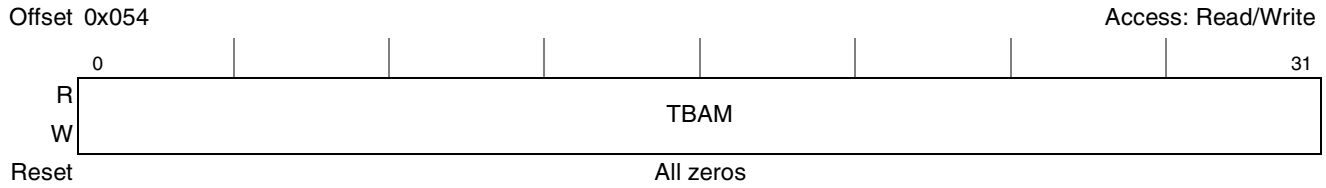


Figure 25-15. Trace Buffer Address Mask Register (TBAMR)

Table 25-20 describes the TBAMR field.

Table 25-20. TBAMR Field Descriptions

Bits	Name	Description
0–31	TBAM	Trace buffer address mask. Low-order bits of the match address mask. A value of zero masks the address comparison for the corresponding address bit.

25.3.2.7 Trace Buffer Transaction Mask Register (TBTMR)

The trace buffer transaction mask register (TBTMR) shown in Figure 25-16 specifies which transaction types to monitor. Each bit in the TBTMR represents a transaction type on the selected interface. The transaction associated with any particular bit depends on the interface being monitored as specified by TBCR1[IFSEL]. Note that the transactions used for defining trace buffer events are the same as those defined for watchpoint monitor events. Thus,Table 25-13 defines the transaction types associated with each interface. Setting a bit enables a hit when this transaction is matched (provided all other match criteria are met and TBCR[TMD] is clear).

Different interfaces support different transaction types, and the same bit may represent different transaction types depending on the interface.

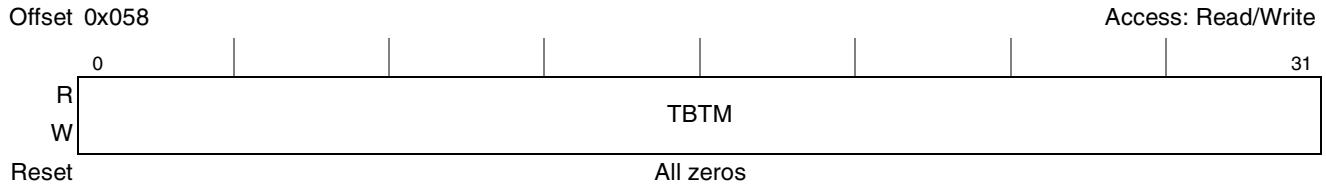


Figure 25-16. Trace Buffer Transaction Mask Register (TBTMR)

Table 25-21 describes the TBTMR field.

Table 25-21. TBTMR Field Descriptions

Bits	Name	Description
0–31	TBTM	Trace buffer transaction mask. Each bit corresponds to a transaction type as defined in Table 25-13. The transaction associated with a bit depends on the interface being monitored. A value of 1 for a given mask bit enables the matching of the transaction associated with that bit. These bits are meaningful only when TBCR0[TMD]=0.

25.3.2.8 Trace Buffer Status Register (TBSR)

The trace buffer status register (TBSR) shown in Figure 25-17 indicates the operational state of the trace buffer.

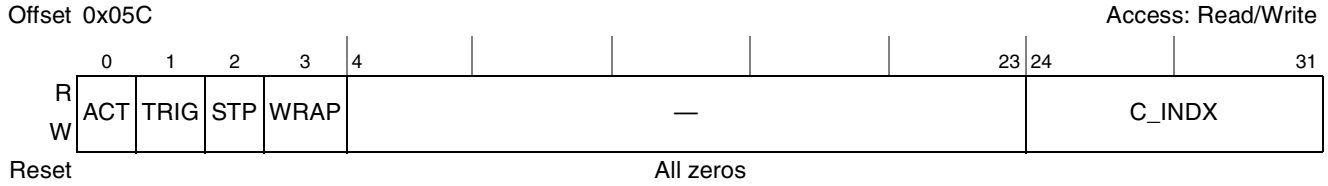


Figure 25-17. Trace Buffer Status Register (TBSR)

Table 25-22 describes the TBSR fields.

Table 25-22. TBSR Field Descriptions

Bits	Name	Description
0	ACT	Active. Indicates trace buffer activity. 0 The start triggering event has not yet occurred. Trace buffer is not armed. 1 The start triggering event has occurred. Trace buffer is armed.
1	TRIG	Triggered. Indicates whether or not a programmed event has been triggered. 0 The programmed event in TBCR0 has not yet been triggered. 1 The programmed event in TBCR0 has been triggered at least once.
2	STP	Stopped. Indicates whether or not a trace buffer stop condition has been detected. 0 No stop condition yet detected. 1 The trace buffer has detected a stop condition and is no longer capturing events.
3	WRAP	Wrapped. Indicates that the trace buffer write pointer has wrapped to the beginning of the buffer at least once. Set when the last entry of the trace buffer is written. 0 Pointer has not yet wrapped. 1 Pointer has wrapped to the beginning at least once.
4–23	—	Reserved
24–31	C_INDX	Current index. Represents the current value of the write pointer at the time TBSR was read. This value may be written by software to initialize the write pointer; however, software is not allowed to write the write pointer while the trace buffer is active. Writes are ignored while the trace buffer is active. It is recommended to write the status register before enabling the trace buffer in order to zero out any bits that might have been set during a prior run and to initialize the write pointer to zero.

25.3.2.9 Trace Buffer Access Control Register (TBACR)

The trace buffer access control register (TBACR), shown in Figure 25-18, enables software to read or write the trace buffer. Each entry is 64 bits; therefore, it takes one write of TBACR and two reads of the access data register (TBADR and TBADHR) to read one 256-entry array entry. Similarly, it takes one write of TBACR and two writes of TBADR and TBADHR to write one array entry. Software can access any entry by writing the appropriate index into TBACR[INDX]. To read or write the buffer sequentially, starting with entry 0, the index must start with a value of 0 and increment every time a new entry is accessed.

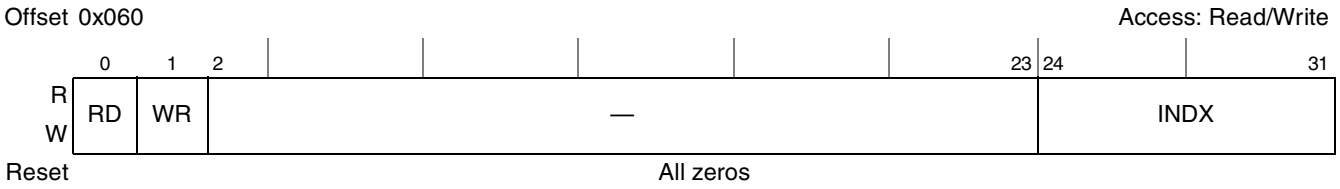


Figure 25-18. Trace Buffer Access Control Register (TBACR)

Table 25-23 describes the TBACR fields.

Table 25-23. TBACR Field Descriptions

Bits	Name	Description
0	RD	Read command. When set, a trace buffer read is performed using the value of TBACR[INDX]. This bit is automatically cleared when the read is performed.
1	WR	Write command. When set, a trace buffer write is performed using the value of TBACR[INDX]. This bit is automatically cleared when the write is performed. A write occurs only if the trace buffer is not active: write requests are ignored while the buffer is active.
2–23	—	Reserved
24–31	INDX	Buffer index to read from or write into (0–255). Used in conjunction with TBACR[RD] and TBACR[WR].

25.3.2.10 Trace Buffer Access Data High Register (TBADHR)

The trace buffer access data high register (TBADHR), shown in Figure 25-19, contains the high-order 32 bits of the data read from the trace buffer during a software-initiated read command (TBACR[RD]), or the write data to be written into the trace buffer during a software-initiated write command (TBACR[WR]). TBACR must be configured to perform a read before this register contains valid data. This register must be initialized by software before configuring the TBACR to perform a write command.

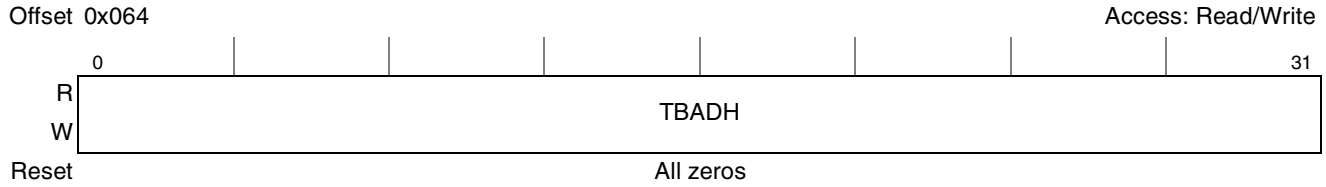


Figure 25-19. Trace Buffer Read High Register (TBADHR)

Table 25-24 describes TBADHR.

Table 25-24. TBADHR Field Descriptions

Bits	Name	Description
0–31	TBADH	Trace buffer access data high. The higher 32 bits of the data read from or to be written into the trace buffer, depending on whether the array is accessed with a read or a write.

25.3.2.11 Trace Buffer Access Data Register (TBADR)

The trace buffer access data register (TBADR), shown in Figure 25-20, contains the low-order 32 bits of the data read from the trace buffer during a software-initiated read command (TBACR[RD]) or the write

data to be written into the trace buffer during a software-initiated write command (TBACR[WR]). TBACR must be configured to perform a read before this register contains valid data. This register must be initialized by software before configuring the TBACR to perform a write command.

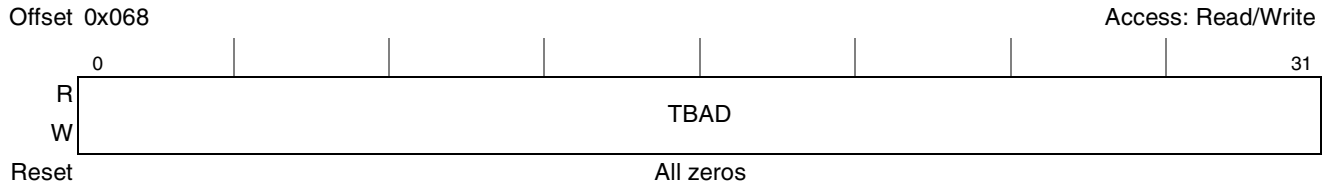


Figure 25-20. Trace Buffer Access Data Register (TBADR)

Table 25-25 describes the TBADR field.

Table 25-25. TBADR Field Descriptions

Bits	Name	Description
0–31	TBAD	Trace buffer access data. Corresponds to the lower 32 bits of the data read from the trace buffer or to be written into the trace buffer, depending on whether software is accessing the array with a read or a write.

25.3.3 Context ID Registers

This section describes the context ID registers. The current context ID register (CCIDR) and programmed context ID registers (PCIDR) are set by software and facilitate debugging complex software.

25.3.3.1 Programmed Context ID Register (PCIDR)

The programmed context ID register (PCIDR), shown in Figure 25-21, contains the user-programmed context ID. This register can be configured to trigger watchpoint events when its value matches the current context ID register (CCIDR), as controlled by WMCR0[ECEN] and WMCR0[NECEN]. See Section 25.3.1.1, “Watchpoint Monitor Control Register 0 (WMCR0),” for more information.

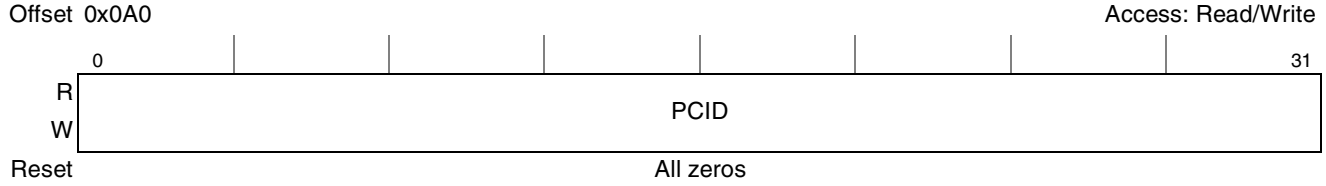


Figure 25-21. Programmed Context ID Register (PCIDR)

Table 25-26 describes the PCIDR field.

Table 25-26. PCIDR Field Descriptions

Bits	Name	Description
0–31	PCID	Programmed context ID. Contains the user-programmed context ID. Compared with current context ID for context-sensitive event triggering.

25.3.3.2 Current Context ID Register (CCIDR)

The current context ID register (CCIDR) shown in Figure 25-22 contains the current context ID. This register is written by software after a context switch and can be used to trigger events when compared with the programmed context ID register (PCIDR).

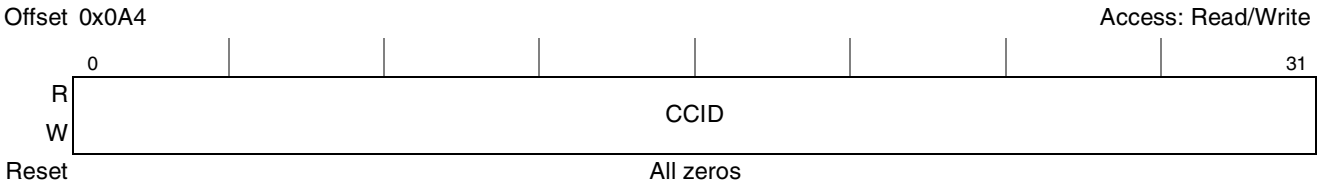


Figure 25-22. Current Context ID Register (CCIDR)

Table 25-27 describes the CCIDR field.

Table 25-27. CCIDR Field Descriptions

Bits	Name	Description
0–31	CCID	Current context ID. Set by user software. Typically loaded immediately following a context switch. Compared with user-programmed context ID for context-sensitive event triggering.

25.3.4 Trigger Out Function

TRIG_OUT provides a convenient mechanism for triggering external system monitors and diagnostic equipment such as logic analyzers. Note that READY is multiplexed with TRIG_OUT. See the last paragraph of Section 4.4.2, “Power-On Reset Sequence” for more information about READY functionality.

When the trace buffer hit is selected by TOSR[SEL], TRIG_OUT is only meaningful if the trace buffer control register 0 (TBCR0) is properly configured to hit on a traceable event. The same holds true for the watchpoint monitor when the watchpoint monitor is selected by TOSR[SEL].

25.3.4.1 Trigger Out Source Register (TOSR)

The trigger out source register (TOSR) shown in Figure 25-23 specifies the source for TRIG_OUT. The three event-trigger sources are the following:

- The watchpoint monitor
- The trace buffer
- The performance monitor

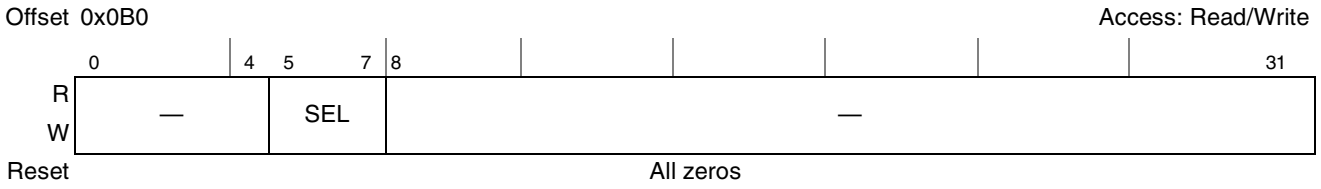


Figure 25-23. Trigger Out Source Register (TOSR)

Table 25-28 describes the TOSR fields.

Table 25-28. TOSR Field Descriptions

Bits	Name	Description
0-4	—	Reserved
5–7	SEL	Select. Selects the source for TRIG_OUT 000 READY signal. Multiplexed with TRIG_OUT. Basic device state indicator. READY asserts whenever the device is not in reset or not asleep. See Chapter 4, "Reset, Clocking, and Initialization," for more details about the reset sequence, and Chapter 23, "Global Utilities," for more information about power management states. 001 Selects the watchpoint monitor hit indication 010 Selects the trace buffer hit indication 011 Selects the performance monitor overflow indication 100–111Reserved
8–31	—	Reserved

25.4 Functional Description

The debug features on the device use the PEX interfaces, the eLBC interface, and the DDR SDRAM interface.

25.4.1 Source and Target IDs

Debug information that is common to all the interfaces is the source ID (SID). The transaction source ID provides enough information to determine which block or port originated a transaction including the distinction between instruction and data fetches from the processor core. Table 25-29 shows the values and interpretation for the 5-bit SID field. Note that the table also includes ports that are only targets, such as the DDR memory controller. These ports are always targets. As such, the value shown represents a target ID (TID) and not a source ID. For ports that can function in both capacities, the value indicates source ID when mastering transactions, and target ID when responding as slave. The TID field is only meaningful when one of the following participates in the transaction:

- The coherency module (MCM) dispatch bus
- The watchpoint monitor (WMCR1[IFSEL] = 000)
- The trace buffer (TBCR1[IFSEL] = 000)

Table 25-29. Source and Target ID Encodings

SID/TID	Source/Target	TID	Source/Target
00000	PCI	10000	e600 instruction fetch—source only
00001	PCI Express 2 (x8)	10001	e600 data access—source only
00010	PCI Express 1 (x4)	10010	Reserved
00011	Reserved	10011	Reserved
00100	Local bus (eLBC)—Target only	10100	Reserved
00101	Reserved	10101	DMA1—source only

Table 25-29. Source and Target ID Encodings (continued)

SID/TID	Source/Target	TID	Source/Target
00110	Reserved	10110	DMA2—source only
00111	Reserved	10111	Reserved
01000	CCSR—target only	11000	Reserved
01001	DIU—source only	11001	Reserved
01010	Boot sequencer—source only	11010	Reserved
01011	Reserved	11011	Reserved
01100	Reserved	11100	Reserved
01101	Reserved	11101	Reserved
01110	Reserved	11110	Reserved
01111	DDR memory controller—target only	11111	Reserved

25.4.2 DDR SDRAM Interface Debug

If MSRCID0 is high when sampled during POR, the debug information from the DDR SDRAM interface is driven on MSRCID[0:4] and MDVAL. This POR value is captured in PORDBGMSR[MEM_SEL] as described in Section 23.4.1.5, "POR Debug Mode Status Register (PORDBGMSR)." In this mode, the source ID appears on MSRCID[0:4] during a RAS or CAS cycle. During any other cycle, the value of MSRCID[0:4] is all ones, which indicates idle cycles on the address/command interface. Similarly, MDVAL is asserted during valid data cycles on the DDR interface.

25.4.3 Enhanced Local Bus Interface Debug

If MSRCID0 is low when sampled during POR, the eLBC is selected as the source for the debug information appearing on MSRCID[0:4] and MDVAL. For more information on this mode, see Section 9.1.3.2, "Source ID Debug Mode."

25.4.4 Watchpoint Monitor

The watchpoint monitor (WM) can be programmed to arm and trigger on many different events including any of the following:

- External event (through TRIG_IN)
- A trace buffer event
- A performance monitor overflow event
- A comparison of the current and programmed context ID registers

A watchpoint event can be used in the following ways:

- Trigger a logic analyzer (using TRIG_OUT)
- Arm or trigger the trace buffer
- Trigger a performance monitor event

The large counters available in the performance monitor block and the interlock between it and the watchpoint monitor support sophisticated debug scenarios.

A WM trigger event may be composed of several events programmed in the watchpoint monitor control registers (WMCR0–WMCR1). Because the watchpoint monitor is disabled by default during POR, these registers must be initialized to make use of this debug feature. Note that the WM address mask register (WMAMR) and the type mask register (WMTMR) are cleared during POR. This means that the watchpoint monitor's default behavior following a power-on reset is to trigger on any address and no transaction type. The reset value of WMCR0[TMD] is 0 which means transaction matching is enabled but since no transaction is selected (WMTMR=0), a match will never occur. Either the transaction matching must be disabled by setting WMCR0[TMD] to a value of 1, or valid transactions must be selected by setting one or more of the WMTMR bits to a value of 1.

25.4.4.1 Watchpoint Monitor—Performance Monitor Events

The WM can produce a performance monitor (PM) event with every trigger. This is accomplished by configuring the performance monitor to count WM events. For more information on this configuration see the events named 'Number of watchpoint monitor hits' and 'Number of trace buffer hits' in Table 24-12.

Multi-level triggers can be created using the watchpoint monitor, the performance monitor, and the trace buffer combined. For example, the WM can be programmed to trigger on events that also increment a PM counter (the performance monitor must also be programmed to respond to this event), the output of which (perfmon_overflow) could trigger the start of tracing in the trace buffer.

25.4.5 Trace Buffer

The trace buffer is a 256×64 array that can capture information about the internal processing of transactions to selected interfaces. The trace buffer controls are a superset of those for the watchpoint monitor. Close inspection of the trace buffer control registers (TBCR*n*) and the WM control registers (WMCR*n*) shows that trace buffer controls not needed for the WM are marked reserved in WMCR*n*. This permits using the trace buffer as a second watchpoint monitor by simply ignoring the trace options.

The trace buffer provides great flexibility about when to start tracing, when to stop tracing, and what to trace. The trace mode field, TBCR0[MODE], indicates when to trace: on every valid cycle, on a watchpoint monitor event, or when all the programmed events in the TBCR are met. This permits a user to program the trace condition in the watchpoint monitor and to program a start or stop condition in the trace buffer control register. The user can also program the TBCR with the conditions in which to stop tracing: on an event, or when the buffer is full. TBCR0[IFSEL] specifies which interface transactions are being captured.

The trace buffer can be programmed to trace the dispatch bus from any of the following:

- Coherency module (MCM)
- Host interface to the DDR controller
- Outbound host interface to the PCI controller
- Outbound host interface to the PCI Express controllers

Transactions come into the MCM, arbitrate for common resources, and get dispatched to the target port. Information such as transaction types, source ID, and other attributes can be captured in any of the selected interfaces.

25.4.5.1 Traced Data Formats (as a Function of TBCR1[IFSEL])

Figure 25-24 shows the trace buffer entry format for an MCM dispatch (CMD) transaction that is specified when TBCR1[IFSEL] = 000.

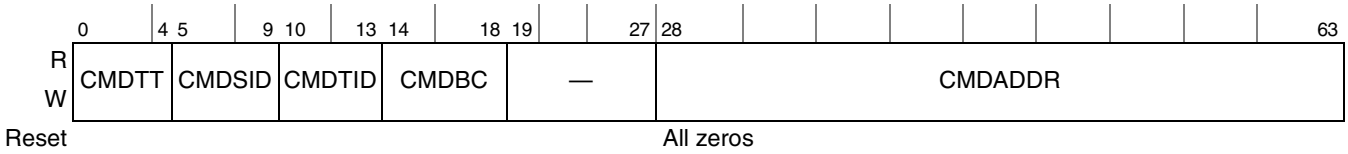


Figure 25-24. Coherency Module Dispatch (CMD) Trace Buffer Entry

Table 25-30 describes the fields of CMD trace buffer entries.

Table 25-30. CMD Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 000)

Bits	Name	Function
0–4	CMDTT	Transaction type. Specifies the transaction type as shown in Table 25-13. For example, a value of zero indicates a write with local processor snoop condition.
5–9	CMDSID	Source ID. Identifies the source of the transaction as shown in Table 25-29. For example, a value of 10101 indicates that DMA is the transaction source.
10–13	CMDTID	Target ID. Identifies the target of the transaction as shown in Table 25-29. For example, a value of 10101 indicates that DMA is the transaction target.
14–18	CMDBC	Byte count. Range: 32 to 1 where a value of 0 indicates 32 bytes. 00000 = 32 bytes 00001 = 1 byte 00010 = 2 bytes . . . 11110 = 30 bytes 11111 = 31 bytes
19–27	—	Reserved
28–63	CMDADDR	Address bits 0–35

Figure 25-25 shows the trace buffer entry format for the DDR SDRAM interface, TBCR1[IFSEL] = 001.

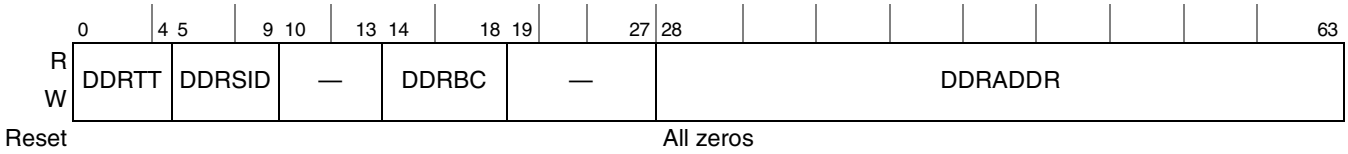


Figure 25-25. DDR Trace Buffer Entry

Table 25-31 describes the fields of DDR SDRAM trace buffer entries when TBCR1[IFSEL] = 001.

Table 25-31. DDR Trace Buffer Entry Field Descriptions (TBCR1[TRSEL] = 001)

Bits	Name	Function
0–4	DDRTT	Transaction type. Specifies the transaction type as shown in Table 25-13. For example, a value of all zeros maps to write.
5–9	DDRSID	Source ID. Specifies the source of the transaction as shown in Table 25-29. For example, a value of 010101 indicates that DMA is the transaction source, and so on.
10–13	—	Reserved
14–18	DDRBC	Byte count
19–27	—	Reserved
28–63	DDRADDR	Address bits 0–35

Figure 25-26 shows the PCI Express trace buffer entry format when TBCR1[IFSEL] = 100 (PEX1 ×4) or TBCR1[IFSEL] = 011 (PEX2 ×8).

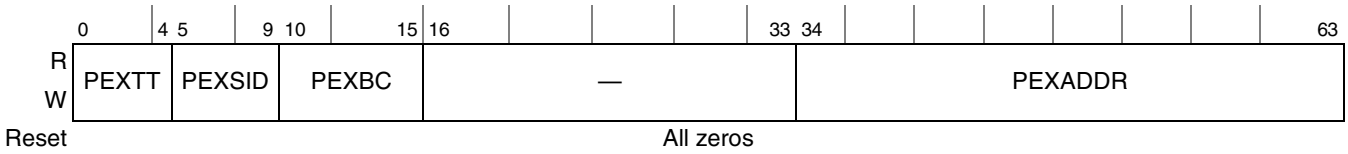


Figure 25-26. PCI Express Trace Buffer Entry

Table 25-32 describes the fields of PCI Express trace buffer entries when TBCR1[IFSEL] = 100 (PEX1 × 4) or TBCR1[IFSEL] = 011 (PEX2 ×8).

Table 25-32. PCI Express Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 100 or 011)

Bits	Name	Function
0–4	PEXTT	Transaction type. Specifies the transaction type as shown in Table 25-13. For example, a value of all zeros maps to write.
5–9	PEXSID	Source ID. Identifies the source of the transaction as shown in Table 25-29.
10–15	PEXBC	Byte count. The size of the transaction. 000000 4 bytes 000001 8 bytes 000010 12 bytes ... 111111 256 bytes
16–33	—	Reserved
34–63	PEXADDR	Address bits 31–2

Figure 25-27 shows the PCI trace buffer entry format when TBCR1[IFSEL] = 010.

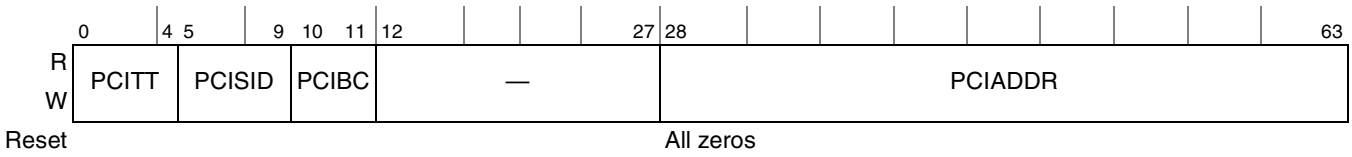


Figure 25-27. PCI Trace Buffer Entry

Table 25-32 describes the fields of PCI trace buffer entries when TBCR1[IFSEL] = 010.

Table 25-33. PCI Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 010)

Bits	Name	Function
0–4	PCITT	Transaction type. Specifies the transaction type as shown in Table 25-13. For example, a value of all zeros maps to write.
5–9	PCISID	Source ID. Identifies the source of the transaction as shown in Table 25-29.
10–11	PCIBC	Byte count. The size of the transaction. 00 32 bytes 01 8 bytes 10 16 bytes 11 24 bytes
12–27	—	Reserved
28–63	PCIADDR	Address bits

25.5 Initialization

Configuring the appropriate control register must be the last step in the initialization sequence for either the watchpoint or trace buffer. That is, all required registers except the corresponding control register must be configured before any control register bits that enable watchpoint or trace events are set.

Appendix A
Complete List of Configuration, Control, and Status Registers

A.1 General Utilities

The general utilities registers are the functional block-specific registers that occupy the first 256 Kbytes of CCSR space (0x0_0000–0x3_FFFF). Each functional block is allocated a 4-Kbyte address range for its registers within the general utilities space.

A.1.1 Local Configuration Control

Table A-1. Local Configuration Control Registers

Local Configuration Control—Block Base Address 0x0_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	CCSRBAR—Configuration, control, and status registers base address register	R/W	0x000F_F700	4.3.1.1.3/4-4
0x008	ALTCBAR—Alternate configuration base address register	R/W	All zeros	4.3.1.2.1/4-5
0x010	ALTCAR—Alternate configuration attribute register	R/W	All zeros	4.3.1.2.2/4-6
0x020	BPTR—Boot page translation register	R/W	All zeros	4.3.1.3.1/4-7

A.1.2 Local Access Windows

Table A-2. Local Access Window Registers

Local Access Windows—Block Base Address 0x0_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xC08	LAWBAR0—Local access window 0 base address register	R/W	All zeros	2.2.1.1/2-5
0xC10	LAWAR0—Local access window 0 attribute register	R/W	All zeros	2.2.1.2/2-5
0xC28	LAWBAR1—Local access window 1 base address register	R/W	All zeros	2.2.1.1/2-5
0xC30	LAWAR1—Local access window 1 attribute register	R/W	All zeros	2.2.1.2/2-5
0xC48	LAWBAR2—Local access window 2 base address register	R/W	All zeros	2.2.1.1/2-5
0xC50	LAWAR2—Local access window 2 attribute register	R/W	All zeros	2.2.1.2/2-5
0xC68	LAWBAR3—Local access window 3 base address register	R/W	All zeros	2.2.1.1/2-5
0xC70	LAWAR3—Local access window 3 attribute register	R/W	All zeros	2.2.1.2/2-5

Table A-2. Local Access Window Registers (continued)

Local Access Windows—Block Base Address 0x0_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xC88	LAWBAR4—Local access window 4 base address register	R/W	All zeros	2.2.1.1/2-5
0xC90	LAWAR4—Local access window 4 attribute register	R/W	All zeros	2.2.1.2/2-5
0xCA8	LAWBAR5—Local access window 5 base address register	R/W	All zeros	2.2.1.1/2-5
0xCB0	LAWAR5—Local access window 5 attribute register	R/W	All zeros	2.2.1.2/2-5
0xCC8	LAWBAR6—Local access window 6 base address register	R/W	All zeros	2.2.1.1/2-5
0xCD0	LAWAR6—Local access window 6 attribute register	R/W	All zeros	2.2.1.2/2-5
0xCE8	LAWBAR7—Local access window 7 base address register	R/W	All zeros	2.2.1.1/2-5
0xCF0	LAWAR7—Local access window 7 attribute register	R/W	All zeros	2.2.1.2/2-5
0xD08	LAWBAR8—Local access window 8 base address register	R/W	All zeros	2.2.1.1/2-5
0xD10	LAWAR8—Local access window 8 attribute register	R/W	All zeros	2.2.1.2/2-5
0xD28	LAWBAR9—Local access window 9 base address register	R/W	All zeros	2.2.1.1/2-5
0xD30	LAWAR9—Local access window 9 attribute register	R/W	All zeros	2.2.1.2/2-5

A.1.3 MPX Coherency Module (MCM)

Table A-3. MCM Registers

MCM—Block Base Address 0x0_1000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	MCM MPX address bus configuration register (ABCR)	R/W	0x0000_0003	7.4.1.1/7-5
0x008	MCM MPX data bus configuration register (DBCR)	R/W	0x0000_0003	7.4.1.2/7-6
0x010	MCM MPX port configuration register (PCR)	R/W	0x0*00_0000	7.4.1.3/7-6
0xBF8	MCM IP block revision register 1	R	0x0101_0200	
0xBFC	MCM IP block revision register 2	R	0x0000_0002	
0xE00	MCM error detect register (EDR)	w1c	All zeros	7.4.1.4/7-7
0xE08	MCM error enable register (EER)	R/W	All zeros	7.4.1.5/7-8
0xE0C	MCM error attributes capture register (EATR)	R	All zeros	7.4.1.6/7-9
0xE10	MCM error low address capture register (ELADR)	R	All zeros	7.4.1.7/7-10
0xE14	MCM error high address capture register (EHADR)	R	All zeros	7.4.1.8/7-10

A.1.4 DDR Memory Controller 1

Table A-4. DDR Memory Controller 1 Registers

DDR Memory Controller 1—Block Base Address 0x0_2000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	CS0_BNDS—Chip select 0 memory bounds	R/W	All zeros	8.4.1.1/8-11
0x008	CS1_BNDS—Chip select 1 memory bounds	R/W	All zeros	8.4.1.1/8-11
0x010	CS2_BNDS—Chip select 2 memory bounds	R/W	All zeros	8.4.1.1/8-11
0x018	CS3_BNDS—Chip select 3 memory bounds	R/W	All zeros	8.4.1.1/8-11
0x080	CS0_CONFIG—Chip select 0 configuration	R/W	All zeros	8.4.1.2/8-11
0x084	CS1_CONFIG—Chip select 1 configuration	R/W	All zeros	8.4.1.2/8-11
0x088	CS2_CONFIG—Chip select 2 configuration	R/W	All zeros	8.4.1.2/8-11
0x08C	CS3_CONFIG—Chip select 3 configuration	R/W	All zeros	8.4.1.2/8-11
0x100	TIMING_CFG_3—DDR SDRAM timing configuration 3	R/W	All zeros	8.4.1.3/8-13
0x104	TIMING_CFG_0—DDR SDRAM timing configuration 0	R/W	0x0011_0105	8.4.1.4/8-14
0x108	TIMING_CFG_1—DDR SDRAM timing configuration 1	R/W	All zeros	8.4.1.5/8-16
0x10C	TIMING_CFG_2—DDR SDRAM timing configuration 2	R/W	All zeros	8.4.1.6/8-18
0x110	DDR_SDRAM_CFG—DDR SDRAM control configuration	R/W	0x0200_0000	8.4.1.7/8-20
0x114	DDR_SDRAM_CFG_2—DDR SDRAM control configuration 2	R/W	All zeros	8.4.1.8/8-23
0x118	DDR_SDRAM_MODE—DDR SDRAM mode configuration	R/W	All zeros	8.4.1.9/8-24
0x11C	DDR_SDRAM_MODE_2—DDR SDRAM mode configuration 2	R/W	All zeros	8.4.1.10/8-25
0x120	DDR_SDRAM_MD_CNTL—DDR SDRAM mode control	R/W	All zeros	8.4.1.11/8-26
0x124	DDR_SDRAM_INTERVAL—DDR SDRAM interval configuration	R/W	All zeros	8.4.1.12/8-28
0x128	DDR_DATA_INIT—DDR SDRAM data initialization	R/W	All zeros	8.4.1.13/8-29
0x130	DDR_SDRAM_CLK_CNTL—DDR SDRAM clock control	R/W	0x0200_0000	8.4.1.14/8-29
0x140–0x144	Reserved	—	—	—
0x148	DDR_INIT_ADDR—DDR training initialization address	R/W	All zeros	8.4.1.15/8-30
0x14C	DDR_INIT_EXT_ADDRESS—DDR training initialization extended address	R/W	All zeros	8.4.1.16/8-30
0x150–0xB1F	Reserved	—	—	—
0xB20	DDRDSR_1—DDR Debug Status Register 1	R	0x*nnn*0_*nnnn*	8.4.1.17/8-31
0xB24	DDRDSR_2—DDR Debug Status Register 2	R	0x*nn*00_0000	8.4.1.18/8-32
0xB28	DDRCDR_1—DDR Control Driver Register 1	R/W	All zeros	8.4.1.19/8-32
0xB2C	DDRCDR_2—DDR Control Driver Register 2	R/W	All zeros	8.4.1.20/8-34
0xB30–0xBF7	Reserved	—	—	—
0xBF8	DDR_IP_REV1—DDR IP block revision 1	R	0x*nnnn_nnnn*[1]	8.4.1.21/8-34

Table A-4. DDR Memory Controller 1 Registers (continued)

DDR Memory Controller 1—Block Base Address 0x0_2000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xBFC	DDR_IP_REV2—DDR IP block revision 2	R	0x00*nn*_00*nn*[1]	8.4.1.22/8-35
0xE00–0xE58	Reserved	—	—	—
0xE00	DATA_ERR_INJECT_HI—Memory data path error injection mask high	R/W	All zeros	8.4.1.23/8-35
0xE04	DATA_ERR_INJECT_LO—Memory data path error injection mask low	R/W	All zeros	8.4.1.24/8-36
0xE08	ECC_ERR_INJECT—Memory data path error injection mask ECC	R/W	All zeros	8.4.1.25/8-36
0xE20	CAPTURE_DATA_HI—Memory data path read capture high	R/W	All zeros	8.4.1.26/8-37
0xE24	CAPTURE_DATA_LO—Memory data path read capture low	R/W	All zeros	8.4.1.27/8-37
0xE28	CAPTURE_ECC—Memory data path read capture ECC	R/W	All zeros	8.4.1.28/8-38
0xE40	ERR_DETECT—Memory error detect	w1c	All zeros	8.4.1.29/8-38
0xE44	ERR_DISABLE—Memory error disable	R/W	All zeros	8.4.1.30/8-39
0xE48	ERR_INT_EN—Memory error interrupt enable	R/W	All zeros	8.4.1.31/8-40
0xE4C	CAPTURE_ATTRIBUTES—Memory error attributes capture	R/W	All zeros	8.4.1.32/8-41
0xE50	CAPTURE_ADDRESS—Memory error address capture	R/W	All zeros	8.4.1.33/8-42
0xE54	CAPTURE_EXT_ADDRESS—Memory error extended address capture	R/W	All zeros	8.4.1.34/8-42
0xE58	ERR_SBE—Single-Bit ECC memory error management	R/W	All zeros	8.4.1.35/8-43

[1] Implementation-dependent reset values are listed in specified section/page.

A.1.5 I²C Controllers

Table A-5. I²C Controller 1 & 2 Registers

I²C Controller 1—Block Base Address 0x0_3000 I²C Controller 2—Block Base Address 0x0_3100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
I²C1 Registers				
0x000	I2CADR—I²C address register	R/W	0x00	12.4.2/12-6
0x004	I2CFDR—I²C frequency divider register	R/W	0x00	12.4.3/12-6

Table A-5. I²C Controller 1 & 2 Registers (continued)

I²C Controller 1—Block Base Address 0x0_3000 I²C Controller 2—Block Base Address 0x0_3100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x008	I2CCR—I²C control register	Mixed	0x00	12.4.4/12-7
0x00C	I2CSR—I²C status register	Mixed	0x81	12.4.5/12-9
0x010	I2CDR—I²C data register	R/W	0x00	12.4.6/12-10
0x014	I2CDFSRR—I²C digital filter sampling rate register	R/W	0x10	12.4.7/12-11
I²C2 Registers				
0x100–0x114	I²C2 Registers[1]			

1 I²C2 has the same memory-mapped registers that are described for I²C1 from 0x000 to 0x014, except the offsets range from 0x100 to 0x114.

A.1.6 DUARTs

Table A-6. DUARTRegisters

DUART1 & DUART2—Block Base Address 0x0_4000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
UART0 Registers				
0x500	URBR—ULCR[DLAB] = 0 UART0 receiver buffer register	R	0x00	13.3.1.1/13-7
0x500	UTHR—ULCR[DLAB] = 0 UART0 transmitter holding register	W	0x00	13.3.1.2/13-7
0x500	UDLB—ULCR[DLAB] = 1 UART0 divisor least significant byte register	R/W	0x00	13.3.1.3/13-8
0x501	UIER—ULCR[DLAB] = 0 UART0 interrupt enable register	R/W	0x00	13.3.1.4/13-9
0x501	UDMB—ULCR[DLAB] = 1 UART0 divisor most significant byte register	R/W	0x00	13.3.1.3/13-8
0x502	UIIR—ULCR[DLAB] = 0 UART0 interrupt ID register	R	0x01	13.3.1.5/13-10
0x502	UFCR—ULCR[DLAB] = 0 UART0 FIFO control register	W	0x00	13.3.1.6/13-12
0x502	UAFR—ULCR[DLAB] = 1 UART0 alternate function register	R/W	0x00	13.3.1.7/13-13
0x503	ULCR—ULCR[DLAB] = x UART0 line control register	R/W	0x00	13.3.1.8/13-14
0x504	UMCR—ULCR[DLAB] = x UART0 modem control register	R/W	0x00	13.3.1.9/13-15
0x505	ULSR—ULCR[DLAB] = x UART0 line status register	R	0x60	13.3.1.10/13-16
0x506	UMSR—ULCR[DLAB] = x UART0 modem status register	R	0x00	13.3.1.11/13-17
0x507	USCR—ULCR[DLAB] = x UART0 scratch register	R/W	0x00	13.3.1.12/13-18
0x510	UDSR—ULCR[DLAB] = x UART0 DMA status register	R	0x01	13.3.1.13/13-19
UART1 Registers				
0x600–0x610	UART1 Registers[1]			

Table A-6. DUARTRegisters (continued)

DUART1 & DUART2—Block Base Address 0x0_4000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
UART2 Registers				
0x700–0x710	UART2 Registers[1]			
UART3 Registers				
0x800–0x810	UART3 Registers[1]			

[1] UART1 has the same memory-mapped registers that are described for UART0 from 0x500 to 0x510, except the offsets range from 0x600 to 0x610. Similarly, the registers for UART2 are located at offsets 0x700 to 0x710 and the registers for UART3 are located at offsets 0x800 to 0x810.

A.1.7 Enhanced Local Bus Controller

Table A-7. 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*	9.3.1.1/9-10
0x008	BR1—Base register 1	R/W	All zeros	9.3.1.1/9-10
0x010	BR2—Base register 2	R/W	All zeros	9.3.1.1/9-10
0x018	BR3—Base register 3	R/W	All zeros	9.3.1.1/9-10
0x020	BR4—Base register 4	R/W	All zeros	9.3.1.1/9-10
0x028	BR5—Base register 5	R/W	All zeros	9.3.1.1/9-10
0x030	BR6—Base register 6	R/W	All zeros	9.3.1.1/9-10
0x038	BR7—Base register 7	R/W	All zeros	9.3.1.1/9-10
0x004	OR0—Options register 0	R/W	0x0000_0FF7	9.3.1.2/9-12
0x00C	OR1—Options register 1	R/W	All zeros	9.3.1.2/9-12
0x014	OR2—Options register 2	R/W	All zeros	9.3.1.2/9-12
0x01C	OR3—Options register 3	R/W	All zeros	9.3.1.2/9-12
0x024	OR4—Options register 4	R/W	All zeros	9.3.1.2/9-12
0x02C	OR5—Options register 5	R/W	All zeros	9.3.1.2/9-12
0x034	OR6—Options register 6	R/W	All zeros	9.3.1.2/9-12
0x03C	OR7—Options register 7	R/W	All zeros	9.3.1.2/9-12
0x040–0x064	Reserved	—	—	—
0x068	MAR—UPM address register	R/W	All zeros	9.3.1.3/9-20

Table A-7. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x06C	Reserved	—	—	—
0x070	MAMR—UPMA mode register	R/W	All zeros	9.3.1.4/9-21
0x074	MBMR—UPMB mode register	R/W	All zeros	9.3.1.4/9-21
0x078	MCMR—UPMC mode register	R/W	All zeros	9.3.1.4/9-21
0x07C–0x080	Reserved	—	—	—
0x084	MRTPR—Memory refresh timer prescaler register	R/W	All zeros	9.3.1.5/9-23
0x088	MDR—UPM/FCM data register	R/W	All zeros	9.3.1.6/9-23
0x08C	Reserved	—	—	—
0x090	LSOR—Special operation initiation register	R/W	All zeros	9.3.1.7/9-24
0x094–0x09C	Reserved	—	—	—
0x0A0	LURT—UPM refresh timer	R/W	All zeros	9.3.1.4/9-21
0x0A4–0x0AC	Reserved	—	—	—
0x0B0	LTESR—Transfer error status register	w1c	All zeros	9.3.1.9/9-26
0x0B4	LTEDR—Transfer error disable register	R/W	All zeros	9.3.1.10/9-28
0x0B8	LTEIR—Transfer error interrupt register	R/W	All zeros	9.3.1.11/9-29
0x0BC	LTEATR—Transfer error attributes register	R/W	All zeros	9.3.1.12/9-30
0x0C0	LTEAR—Transfer error address register	R/W	All zeros	9.3.1.13/9-31
0x0C4–0x0CC	Reserved	—	—	—
0x0D0	LBCR—Configuration register	R/W	All zeros	9.3.1.14/9-32
0x0D4	LCRR—Clock ratio register	R/W	0x8000_0008	9.3.1.15/9-33
0x0D8–0x0DC	Reserved	—	—	—
0x0E0	FMR—Flash mode register	R/W	0x0000_0*n*00	9.3.1.16/9-34
0x0E4	FIR—Flash instruction register	R/W	All zeros	9.3.1.17/9-36
0x0E8	FCR—Flash command register	R/W	All zeros	9.3.1.18/9-37
0x0EC	FBAR—Flash block address register	R/W	All zeros	9.3.1.19/9-38
0x0F0	FPAR—Flash page address register	R/W	All zeros	9.3.1.20/9-38
0x0F4	FBCR—Flash byte count register	R/W	All zeros	9.3.1.21/9-40

Table A-7. Enhanced Local Bus Controller Registers (continued)

Enhanced Local Bus Controller—Block Base Address 0x0_5000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0F8–0x0FC	Reserved	—	—	—
0x110–0xFFC	Reserved	—	—	—

A.1.8 Serial Peripheral Interface (SPI)

Table A-8. Serial Peripheral Interface Registers

Serial Peripheral Interface—Block Base Address 0x0_7000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000–0x01F	Reserved	—	—	—
0x020	SPI mode register (SPMODE)	R/W	All zeros	15.4.1.1/1515-9
0x024	SPI event register (SPIE)	Mixed	All zeros	15.4.1.2/1515-12
0x028	SPI mask register (SPIM)	R/W	All zeros	15.4.1.3/1515-13
0x02C	SPI command register (SPCOM)	W	All zeros	15.4.1.4/1515-14
0x030	SPI transmit register (SPITD)	W	All zeros	15.4.1.5/1515-14
0x034	SPI receive register (SPIRD)	R	0xFFFF_FFFF	15.4.1.6/1515-15
0x038–0xFFF	Reserved	—	—	—

A.1.9 PCI Controller

Table A-9. PCI Controller Registers

PCI Controller—Block Base Address 0x0_8000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
PCI Configuration Access Registers				
0x000	CFG_ADDR—PCI configuration address	R/W	All zeros	20.3.1.1.1/20-14
0x004	CFG_DATA—PCI configuration data	R/W	All zeros	20.3.1.1.2/20-15
0x008	INT_ACK—PCI interrupt acknowledge	R	All zeros	20.3.1.1.3/20-15
0x00C–0xBFC	Reserved	—	—	—
PCI ATMU Registers—Outbound and Inbound				
0xC00–0xC3C–Outbound Window 0 (default)				
0xC00	POTAR0—PCI outbound window 0 (default) translation address register	R/W	All zeros	20.3.1.2.1/20-16

Table A-9. PCI Controller Registers (continued)

PCI Controller—Block Base Address 0x0_8000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xC04	POTEAR0—PCI outbound window 0 (default) translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC08	Reserved	—	—	
0xC0C	Reserved	—	—	
0xC10	POWAR0—PCI outbound window 0 (default) attributes register	R/W	0x8004_401F	20.3.1.2.4/20-17
0xC14–0xC1C	Reserved	—	—	
0xC20–0xC3C—Outbound Window 1				
0xC20	POTAR1—PCI outbound window 1 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC24	POTEAR1—PCI outbound window 1 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC28	POWBAR1—PCI outbound window 1 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC2C	Reserved	—	—	
0xC30	POWAR1—PCI outbound window 1 attributes register	R/W	All zeros	20.3.1.2.4/20-17
0xC34–0xC3C	Reserved	—	—	
0xC40–0xC5C—Outbound Window 2				
0xC40	POTAR2—PCI outbound window 2 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC44	POTEAR2—PCI outbound window 2 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC48	POWBAR2—PCI outbound window 2 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC4C	Reserved	—	—	
0xC50	POWAR2—PCI outbound window 2 attributes register	R/W	All zeros	20.3.1.2.4/20-17
0xC54–0xC5C	Reserved	—	—	
0xC60–0xC7C—Outbound Window 3				
0xC60	POTAR3—PCI outbound window 3 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC64	POTEAR3—PCI outbound window 3 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC68	POWBAR3—PCI outbound window 3 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC6C	Reserved	—	—	
0xC70	POWAR3—PCI outbound window 3 attributes register	R/W	All zeros	20.3.1.2.4/20-17
0xC74–0xC7C	Reserved	—	—	
0xC80–0xC9C—Outbound Window 4				

Table A-9. PCI Controller Registers (continued)

PCI Controller—Block Base Address 0x0_8000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xC80	POTAR4—PCI outbound window 4 translation address register	R/W	All zeros	20.3.1.2.1/20-16
0xC84	POTEAR4—PCI outbound window 4 translation extended address register	R/W	All zeros	20.3.1.2.2/20-16
0xC88	POWBAR4—PCI outbound window 4 base address register	R/W	All zeros	20.3.1.2.3/20-17
0xC8C	Reserved	—	—	
0xC90	POWAR4—PCI outbound window 4 attributes register	R/W	All zeros	20.3.1.2.4/20-17
0xC94–0xD9C	Reserved	—	—	
0xDA0–0xDBC–Inbound Window 3				
0xDA0	PITAR3—PCI inbound window 3 translation address register	R/W	All zeros	20.3.1.3.1/20-20
0xDA4	Reserved	—	—	
0xDA8	PIWBAR3—PCI inbound window 3 base address register	R/W	All zeros	20.3.1.3.2/20-20
0xDAC	PIWBEAR3—PCI inbound window 3 base extended address register	R/W	All zeros	20.3.1.3.3/20-21
0xDB0	PIWAR3—PCI inbound window 3 attributes register	R/W	All zeros	20.3.1.3.4/20-21
0xDB4–0xDBC	Reserved	—	—	
0xDC0–0xDDC–Inbound Window 2				
0xDC0	PITAR2—PCI inbound window 2 translation address register	R/W	All zeros	20.3.1.3.1/20-20
0xDC4	Reserved	—	—	
0xDC8	PIWBAR2—PCI inbound window 2 base address register	R/W	All zeros	20.3.1.3.2/20-20
0xDCC	PIWBEAR2—PCI inbound window 2 base extended address register	R/W	All zeros	20.3.1.3.3/20-21
0xDD0	PIWAR2—PCI inbound window 2 attributes register	R/W	All zeros	20.3.1.3.4/20-21
0xDD4–0xDDC	Reserved	—	—	
0xDE0–0xDFC–Inbound Window 1				
0xDE0	PITAR1—PCI inbound window 1 translation address register	R/W	All zeros	20.3.1.3.1/20-20
0xDE4	Reserved	—	—	
0xDE8	PIWBAR1—PCI inbound window 1 base address register	R/W	All zeros	20.3.1.3.2/20-20
0xDEC	Reserved	—	—	
0xDF0	PIWAR1—PCI inbound window 1 attributes register	R/W	All zeros	20.3.1.3.4/20-21
0xDF4–0xDFC	Reserved	—	—	
PCI Error Management Registers				
0xE00	ERR_DR—PCI error detect register	w1c	All zeros	20.3.1.4.1/20-24
0xE04	ERR_CAP_DR—PCI error capture disabled register	R/W	All zeros	20.3.1.4.2/20-25

Table A-9. PCI Controller Registers (continued)

PCI Controller—Block Base Address 0x0_8000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xE08	ERR_EN—PCI error enable register	R/W	All zeros	20.3.1.4.3/20-26
0xE0C	ERR_ATTRIB—PCI error attributes capture register	R/W	All zeros	20.3.1.4.4/20-27
0xE10	ERR_ADDR—PCI error address capture register	R/W	All zeros	20.3.1.4.5/20-28
0xE14	ERR_EXT_ADDR—PCI error extended address capture register	R/W	All zeros	20.3.1.4.6/20-28
0xE18	ERR_DL—PCI error data low capture register	R/W	All zeros	20.3.1.4.7/20-29
0xE1C	ERR_DH—PCI error data high capture register	R/W	All zeros	20.3.1.4.8/20-29
0xE20	GAS_TIMR—PCI gasket timer register	R/W	0x0100_3FFF	20.3.1.4.9/20-29
0xE28–0xEFC	Reserved	—	—	
0xF00–0xFFC	Reserved for debug	—	—	

A.1.10 PCI Express Controllers

Table A-10. PCI Express Controller 1 & 2 Registers

PCI Express Controller 2 (×8)—Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
PCI Express Configuration Access Registers				
0x000	PEX_CONFIG_ADDR—PCI Express configuration address register	R/W	All zeros	21.3.2.1/21-10
0x004	PEX_CONFIG_DATA—PCI Express configuration data register	R/W	All zeros	21.3.2.2/21-10
0x008	Reserved	—	—	
0x00C	PEX_OTB_CPL_TOR—PCI Express outbound completion timeout register	R/W	0x0010_FFFF	21.3.2.3/21-11
0x010	PEX_CONF_RTY_TOR—PCI Express configuration retry timeout register	R/W	0x0400_FFFF	21.3.2.4/21-12
0x014	PEX_CONFIG—PCI Express configuration register	R/W	All zeros	21.3.2.5/21-12
0x018–0x01C	Reserved	—	—	
PCI Express Power Management Event & Message Registers				
0x020	PEX_PME_MES_DR—PCI Express PME & message detect register	w1c	All zeros	21.3.3.1/21-13
0x024	PEX_PME_MES_DISR—PCI Express PME & message disable register	R/W	All zeros	21.3.3.2/21-15
0x028	PEX_PME_MES_IER—PCI Express PME & message interrupt enable register	R/W	All zeros	21.3.3.3/21-16

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

PCI Express Controller 2 (×8)—Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x02C	PEX_PMCR—PCI Express power management command register	R/W	All zeros	21.3.3.4/21-18
0x030–0xBF4	Reserved	—	—	
PCI Express IP Block Revision Registers				
0xBF8	IP block revision register 1 (PEX_IP_BLK_REV1)	R	0x0208_0100	21.3.4.1/21-19
0xBFC	IP block revision register 2 (PEX_IP_BLK_REV2)	R	All zeros	21.3.4.2/21-19
PCI Express ATMU Registers				
Outbound Window 0 (Default)				
0xC00	PEXOTAR0—PCI Express outbound translation address register 0 (default)	R/W	All zeros	21.3.5.1.1/21-20
0xC04	PEXOTEAR0—PCI Express outbound translation extended address register 0 (default)	R/W	All zeros	21.3.5.1.2/21-21
0xC08–0xC0C	Reserved	—	—	
0xC10	PEXOWAR0—PCI Express outbound window attributes register 0 (default)	Mixed	0x8004_4023	21.3.5.1.4/21-22
0xC14–0xC1C	Reserved	—	—	
Outbound Window 1				
0xC20	PEXOTAR1—PCI Express outbound translation address register 1	R/W	All zeros	21.3.5.1.1/21-20
0xC24	PEXOTEAR1—PCI Express outbound translation extended address register 1	R/W	All zeros	21.3.5.1.2/21-21
0xC28	PEXOWBAR1—PCI Express outbound window base address register 1	R/W	All zeros	21.3.5.1.3/21-21
0xC2C	Reserved	—	—	
0xC30	PEXOWAR1—PCI Express outbound window attributes register 1	R/W	0x0004_4023	21.3.5.1.4/21-22
0xC34–0xC3C	Reserved	—	—	
Outbound Window 2				
0xC40	PEXOTAR2—PCI Express outbound translation address register 2	R/W	All zeros	21.3.5.1.1/21-20
0xC44	PEXOTEAR2—PCI Express outbound translation extended address register 2	R/W	All zeros	21.3.5.1.2/21-21
0xC48	PEXOWBAR2—PCI Express outbound window base address register 2	R/W	All zeros	21.3.5.1.3/21-21
0xC4C	Reserved	—	—	

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

PCI Express Controller 2 (×8)—Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xC50	PEXOWAR2—PCI Express outbound window attributes register 2	R/W	0x0004_4023	21.3.5.1.4/21-22
0xC54–0xC5C	Reserved	—	—	
Outbound Window 3				
0xC60	PEXOTAR3—PCI Express outbound translation address register 3	R/W	All zeros[1]	21.3.5.1.1/21-20
0xC64	PEXOTEAR3—PCI Express outbound translation extended address register 3	R/W	All zeros	21.3.5.1.2/21-21
0xC68	PEXOWBAR3—PCI Express outbound window base address register 3	R/W	All zeros[2]	21.3.5.1.3/21-21
0xC6C	Reserved	—	—	
0xC70	PEXOWAR3—PCI Express outbound window attributes register 3	R/W	All zeros[3]	21.3.5.1.4/21-22
0xC74–0xC7C	Reserved	—	—	
Outbound Window 4				
0xC80	PEXOTAR4—PCI Express outbound translation address register 4	R/W	All zeros	21.3.5.1.1/21-20
0xC84	PEXOTEAR4—PCI Express outbound translation extended address register 4	R/W	All zeros	21.3.5.1.2/21-21
0xC88	PEXOWBAR4—PCI Express outbound window base address register 4	R/W	All zeros	21.3.5.1.3/21-21
0xC8C	Reserved	—	—	
0xC90	PEXOWAR4—PCI Express outbound window attributes register 4	R/W	0x0004_4023	21.3.5.1.4/21-22
0xC94–0xC9C	Reserved	—	—	
0xD14–0xD9C	Reserved	—	—	
Inbound Window 3				
0xDA0	PEXITAR3—PCI Express inbound translation address register 3	R/W	All zeros	21.3.5.2.3/21-26
0xDA4	Reserved	—	—	
0xDA8	PEXIWBAR3—PCI Express inbound window base address register 3	R/W	All zeros	21.3.5.2.4/21-27
0xDAC	PEXIWBEAR3—PCI Express inbound window base extended address register 3	R/W	All zeros	21.3.5.2.5/21-27
0xDB0	PEXIWAR3—PCI Express inbound window attributes register 3	R/W	0x20F4_4023	21.3.5.2.6/21-28

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

PCI Express Controller 2 (×8)—Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xDB4– 0xDBC	Reserved	—	—	
Inbound Window 2				
0xDC0	PEXITAR2—PCI Express inbound translation address register 2	R/W	All zeros	21.3.5.2.3/21-26
0xDC4	Reserved	—	—	
0xDC8	PEXIWBAR2—PCI Express inbound window base address register 2	R/W	All zeros	21.3.5.2.4/21-27
0xDCC	PEXIWBEAR2—PCI Express inbound window base extended address register 2	R/W	All zeros	21.3.5.2.5/21-27
0xDD0	PEXIWAR2—PCI Express inbound window attributes register 2	R/W	0x20F4_4023	21.3.5.2.6/21-28
0xDD4– 0xDDC	Reserved	—	—	
Inbound Window 1				
0xDE0	PEXITAR1—PCI Express inbound translation address register 1	R/W	All zeros	21.3.5.2.3/21-26
0xDE4	Reserved	—	—	
0xDE8	PEXIWBAR1—PCI Express inbound window base address register 1	R/W	All zeros	21.3.5.2.4/21-27
0xDEC	Reserved	—	—	
0xDF0	PEXIWAR1—PCI Express inbound window attributes register 1	R/W	0x20F4_4023	21.3.5.2.6/21-28
0xDF4– 0xDFC	Reserved	—	—	
PCI Express Error Management Registers				
0xE00	PEX_ERR_DR—PCI Express error detect register	w1c	All zeros	21.3.6.1/21-30
0xE04	Reserved	—	—	—
0xE08	PEX_ERR_EN—PCI Express error interrupt enable register	R/W	All zeros	21.3.6.2/21-33
0xE0C	Reserved	—	—	—
0xE10	PEX_ERR_DISR—PCI Express error disable register	R/W	All zeros	21.3.6.3/21-35
0xE14– 0xE1C	Reserved	—	—	—
0xE20	PEX_ERR_CAP_STAT—PCI Express error capture status register	Mixed	All zeros	21.3.6.4/21-36
0xE24	Reserved	—	—	—
0xE28	PEX_ERR_CAP_R0—PCI Express error capture register 0	R/W	All zeros	21.3.6.5/21-37
0xE2C	PEX_ERR_CAP_R1—PCI Express error capture register 1	R/W	All zeros	21.3.6.6/21-39
0xE30	PEX_ERR_CAP_R2—PCI Express error capture register 2	R/W	All zeros	21.3.6.7/21-40

Table A-10. PCI Express Controller 1 & 2 Registers (continued)

PCI Express Controller 2 (×8)—Block Base Address 0x0_9000 PCI Express Controller 1 (×4)—Block Base Address 0x0_A000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0xE34	PEX_ERR_CAP_R3—PCI Express error capture register 3	R/W	All zeros	21.3.6.8/21-42
0xE38– 0xFFC	Reserved	—	—	
PCI Express Controller 1 (×4) Memory-Mapped Registers				
0x000– 0xFFC	PCI Express Controller 1 (×4) registers **Note:** All registers defined for PCI Express Controller 2 (×8) are also defined for PCI Express Controller 1 (×4); the offsets of PCI Express Controller 1 (×4) registers are the same except they have a different block base address.			

1 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOTAR3 is 0x000F_FFF0.

2 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOWBAR3 is 0x000F_FF00.

3 If the device is configured to use the alternate boot vector (cfg_boot_vec = 0), the reset value for PCI controller 1 PEXOWAR3 is 0x8004_400F.

A.1.11 DMA Controller 2

Table A-11. DMA Controller 2 Registers

DMA Controller 2—Block Base Address 0x0_C000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x100	MR0—DMA 0 mode register	R/W	All zeros	19.3.1.1/19-9
0x104	SR0—DMA 0 status register	Mixed	All zeros	19.3.1.2/19-12
0x108	ECLNDAR0—DMA 0 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x10C	CLNDAR0—DMA 0 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x110	SATR0—DMA 0 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x114	SAR0—DMA 0 source address register	R/W	All zeros	19.3.1.5/19-16
0x118	DATR0—DMA 0 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x11C	DAR0—DMA 0 destination address register	R/W	All zeros	19.3.1.7/19-18
0x120	BCR0—DMA 0 byte count register	R/W	All zeros	19.3.1.8/19-19
0x124	ENLNDAR0—DMA 0 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x128	NLNDAR0—DMA 0 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x130	ECLSDAR0—DMA 0 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x134	CLSDAR0—DMA 0 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20

Table A-11. DMA Controller 2 Registers (continued)

DMA Controller 2—Block Base Address 0x0_C000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x138	ENLSDAR0—DMA 0 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x13C	NLSDAR0—DMA 0 next list descriptor address register	Mixed	All zeros	19.3.1.11/19-22
0x140	SSR0—DMA 0 source stride register	R/W	All zeros	19.3.1.12/19-23
0x144	DSR0—DMA 0 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x148–0x17C	Reserved	—	—	—
0x180	MR1—DMA 1 mode register	R/W	All zeros	19.3.1.1/19-9
0x184	SR1—DMA 1 status register	Mixed	All zeros	19.3.1.2/19-12
0x188	ECLNDAR1—DMA 1 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x18C	CLNDAR1—DMA 1 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x190	SATR1—DMA 1 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x194	SAR1—DMA 1 source address register	R/W	All zeros	19.3.1.5/19-16
0x198	DATR1—DMA 1 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x19C	DAR1—DMA 1 destination address register	R/W	All zeros	19.3.1.7/19-18
0x1A0	BCR1—DMA 1 byte count register	R/W	All zeros	19.3.1.8/19-19
0x1A4	ENLNDAR1—DMA 1 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x1A8	NLNDAR1—DMA 1 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x1B0	ECLSDAR1—DMA 1 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x1B4	CLSDAR1—DMA 1 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x1B8	ENLSDAR1—DMA 1 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x1BC	NLSDAR1—DMA 1 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22
0x1C0	SSR1—DMA 1 source stride register	R/W	All zeros	19.3.1.12/19-23
0x1C4	DSR1—DMA 1 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x1C8–0x1FC	Reserved	—	—	—
0x200	MR2—DMA 2 mode register	R/W	All zeros	19.3.1.1/19-9
0x204	SR2—DMA 2 status register	Mixed	All zeros	19.3.1.2/19-12
0x208	ECLNDAR2—DMA 2 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x20C	CLNDAR2—DMA 2 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x210	SATR2—DMA 2 source attributes register	R/W	All zeros	19.3.1.4/19-15

Table A-11. DMA Controller 2 Registers (continued)

DMA Controller 2—Block Base Address 0x0_C000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x214	SAR2—DMA 2 source address register	R/W	All zeros	19.3.1.5/19-16
0x218	DATR2—DMA 2 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x21C	DAR2—DMA 2 destination address register	R/W	All zeros	19.3.1.7/19-18
0x220	BCR2—DMA 2 byte count register	R/W	All zeros	19.3.1.8/19-19
0x224	ENLNDAR2—DMA 2 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x228	NLNDAR2—DMA 2 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x230	ECLSDAR2—DMA 2 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x234	CLSDAR2—DMA 2 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x238	ENLSDAR2—DMA 2 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x23C	NLSDAR2—DMA 2 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22
0x240	SSR2—DMA 2 source stride register	R/W	All zeros	19.3.1.12/19-23
0x244	DSR2—DMA 2 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x248–0x27C	Reserved	—	—	—
0x280	MR3—DMA 3 mode register	R/W	All zeros	19.3.1.1/19-9
0x284	SR3—DMA 3 status register	Mixed	All zeros	19.3.1.2/19-12
0x288	ECLNDAR3—DMA 3 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x28C	CLNDAR3—DMA 3 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x290	SATR3—DMA 3 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x294	SAR3—DMA 3 source address register	R/W	All zeros	19.3.1.5/19-16
0x298	DATR3—DMA 3 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x29C	DAR3—DMA 3 destination address register	R/W	All zeros	19.3.1.7/19-18
0x2A0	BCR3—DMA 3 byte count register	R/W	All zeros	19.3.1.8/19-19
0x2A4	ENLNDAR3—DMA 3 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x2A8	NLNDAR3—DMA 3 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x2B0	ECLSDAR3—DMA 3 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x2B4	CLSDAR3—DMA 3 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x2B8	ENLSDAR3—DMA 3 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x2BC	NLSDAR3—DMA 3 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22

Table A-11. DMA Controller 2 Registers (continued)

DMA Controller 2—Block Base Address 0x0_C000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x2C0	SSR3—DMA 3 source stride register	R/W	All zeros	19.3.1.12/19-23
0x2C4	DSR3—DMA 3 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x2C8–0x2FC	Reserved	—	—	—
0x300	DGSR—DMA general status register	R	All zeros	19.3.1.14/19-24

A.1.12 GPIO Controllers

Table A-12. GPIO Controller Registers

GPIO Controller 1—Block Base Address 0x0_F000 GPIO Controller 2—Block Base Address 0x0_F100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	GPIO1 Direction Register (GPDIR)	R/W	All zeros	22.5.1/22-4
0x004	GPIO1 Open Drain Register (GPODR)	R/W	All zeros	22.5.2/22-5
0x008	GPIO1 Data register (GPDAT)	R/W	All zeros	22.5.3/22-5
0x00C	GPIO1 Interrupt Event Register (GPIER)	R/W	Undefined	22.5.4/22-6
0x010	GPIO1 Interrupt Mask Register (GPIMR)	R/W	All zeros	22.5.5/22-7
0x014	GPIO1 External Interrupt Control Register (GPICR)	R/W	All zeros	22.5.6/22-7
0x100–0x114	GPIO2 Registers—Same as GPIO1 but offset by 0x100			

A.1.13 Synchronous Serial Interface Controllers

Table A-13. SSI Controller Registers

SSI Controller 1—Block Base Address 0x1_6000 SSI Controller 2—Block Base Address 0x1_6100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
000	STX0—SSI Transmit Data Register 0	R/W	All zeros	16.3.1.1/16-25
004	STX1—SSI Transmit Data Register 1	R/W	All zeros	
008	SRX0—SSI Receive Data Register 0	Read-only	All zeros	16.3.1.4/16-28
00C	SRX1—SSI Receive Data Register 1	Read-only	All zeros	
010	SCR—SSI Control Register	R/W	All zeros	16.3.1.7/16-31
014	SISR—SSI Interrupt Status Register	Mixed	0x0000_3003	16.3.1.8/16-33
018	SIER—SSI Interrupt Enable Register	R/W	0x0000_3003	16.3.1.9/16-38
01C	STCR—SSI Transmit Configuration Register	R/W	0x0000_0200	16.3.1.10/16-39
020	SRCR—SSI Receive Configuration Register	R/W	0x0000_0200	16.3.1.11/16-41

Table A-13. SSI Controller Registers (continued)

SSI Controller 1—Block Base Address 0x1_6000 SSI Controller 2—Block Base Address 0x1_6100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
024	STCCR—SSI Transmit Clock Control Register	R/W	0x0004_0000	16.3.1.12/16-42
028	SRCCR—SSI Receive Clock Control Register	R/W	0x0004_0000	
02C	SFCSR—SSI FIFO Control/Status Register	R/W	0x0081_0081	16.3.1.13/16-45
038	SACNT—SSI AC97 Control Register	R/W	All zeros	16.3.1.14/16-48
03C	SACADD—SSI AC97 Command Address Register	R/W	All zeros	16.3.1.15/16-49
040	SACDAT—SSI AC97 Command Data Register	R/W	All zeros	16.3.1.16/16-50
044	SATAG—SSI AC97 Tag Register	R/W	All zeros	16.3.1.17/16-50
048	STMSK—SSI Transmit Time Slot Mask Register	R/W	All zeros	16.3.1.18/16-51
04C	SRMSK—SSI Receive Time Slot Mask Register	R/W	All zeros	16.3.1.19/16-51
050	SACCST—SSI AC97 Channel Status Register	R	All zeros	16.3.1.20/16-52
054	SACCEN—SSI AC97 Channel Enable Register	W	All zeros	16.3.1.21/16-52
058	SACCDIS—SSI AC97 Channel Disable Register	W	All zeros	16.3.1.22/16-53
100–158	SSI2 Registers[1]			

[1] SSI2 has the same memory-mapped registers that are described for SSI1 from 0x000 to 0x058, except the offsets range from 0x100 to 0x158.

A.1.14 Global Timer Modules

Table A-14. Global Timer Module Registers

Global Timer Module 1—Block Base Address 0x1_7000 Global Timer Module 2—Block Base Address 0x1_7100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x00	Timer 1 and 2 global timers configuration register (GTCFR1)	R/W	0x00	17.3.1/17-7
0x01–0x03	Reserved	—	—	—
0x04	Timer 3 and 4 global timers configuration register (GTCFR2)	R/W	0x00	17.3.1/17-7
0x05–0x0f	Reserved	—	—	—
0x10	Timer 1 global timers mode register (GTMDR1)	R/W	0x0000	17.3.2/17-10
0x12	Timer 2 global timers mode register (GTMDR2)			
0x14	Timer 1 global timers reference register (GTRFR1)	R/W	0x0000	17.3.3/17-11
0x16	Timer 2 global timers reference register (GTRFR2)			
0x18	Timer 1 global timers capture register (GTCPR1)	R/W	0x0000	17.3.4/17-12
0x1A	Timer 2 global timers capture register (GTCPR2)			
0x1C	Timer 1 global timers counter register (GTCNR1)	R/W	0x0000	17.3.5/17-12
0x1E	Timer 2 global timers counter register (GTCNR2)			

Table A-14. Global Timer Module Registers (continued)

Global Timer Module 1—Block Base Address 0x1_7000 Global Timer Module 2—Block Base Address 0x1_7100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x20	Timer 3 global timers mode register (GTMDR3)	R/W	0x0000	17.3.2/17-10
0x22	Timer 4 global timers mode register (GTMDR4)			
0x24	Timer 3 global timers reference register (GTRFR3)	R/W	0x0000	17.3.3/17-11
0x26	Timer 4 global timers reference register (GTRFR4)			
0x28	Timer 3 global timers capture register (GTCPR3)	R	0x0000	17.3.4/17-12
0x2A	Timer 4 global timers capture register (GTCPR4)			
0x2C	Timer 3 global timers counter register (GTCNR3)	R/W	0x0000	17.3.5/17-12
0x2E	Timer 4 global timers counter register (GTCNR4)			
0x30	Timer 1 global timers event register (GTEVR1)	Special	0x0000	17.3.6/17-13
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	17.3.7/17-13
0x3A	Timer 2 global timers prescale register (GTPSR2)			
0x3C	Timer 3 global timers prescale register (GTPSR3)			
0x3E	Timer 4 global timers prescale register (GTPSR4)			
0x00–0x3E	GTM2 Registers—Same as GTM1 registers but offset by 0x100			

A.1.15 DMA Controller 1

Table A-15. DMA Controller 1 Registers

DMA Controller 1—Block Base Address 0x2_1000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x100	MR0—DMA 0 mode register	R/W	All zeros	19.3.1.1/19-9
0x104	SR0—DMA 0 status register	Mixed	All zeros	19.3.1.2/19-12
0x108	ECLNDAR0—DMA 0 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x10C	CLNDAR0—DMA 0 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x110	SATR0—DMA 0 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x114	SAR0—DMA 0 source address register	R/W	All zeros	19.3.1.5/19-16
0x118	DATR0—DMA 0 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x11C	DAR0—DMA 0 destination address register	R/W	All zeros	19.3.1.7/19-18
0x120	BCR0—DMA 0 byte count register	R/W	All zeros	19.3.1.8/19-19

Table A-15. DMA Controller 1 Registers (continued)

DMA Controller 1—Block Base Address 0x2_1000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x124	ENLNDAR0—DMA 0 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x128	NLNDAR0—DMA 0 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x130	ECLSDAR0—DMA 0 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x134	CLSDAR0—DMA 0 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x138	ENLSDAR0—DMA 0 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x13C	NLSDAR0—DMA 0 next list descriptor address register	Mixed	All zeros	19.3.1.11/19-22
0x140	SSR0—DMA 0 source stride register	R/W	All zeros	19.3.1.12/19-23
0x144	DSR0—DMA 0 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x148–0x17C	Reserved	—	—	—
0x180	MR1—DMA 1 mode register	R/W	All zeros	19.3.1.1/19-9
0x184	SR1—DMA 1 status register	Mixed	All zeros	19.3.1.2/19-12
0x188	ECLNDAR1—DMA 1 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x18C	CLNDAR1—DMA 1 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x190	SATR1—DMA 1 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x194	SAR1—DMA 1 source address register	R/W	All zeros	19.3.1.5/19-16
0x198	DATR1—DMA 1 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x19C	DAR1—DMA 1 destination address register	R/W	All zeros	19.3.1.7/19-18
0x1A0	BCR1—DMA 1 byte count register	R/W	All zeros	19.3.1.8/19-19
0x1A4	ENLNDAR1—DMA 1 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x1A8	NLNDAR1—DMA 1 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x1B0	ECLSDAR1—DMA 1 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x1B4	CLSDAR1—DMA 1 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x1B8	ENLSDAR1—DMA 1 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x1BC	NLSDAR1—DMA 1 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22
0x1C0	SSR1—DMA 1 source stride register	R/W	All zeros	19.3.1.12/19-23
0x1C4	DSR1—DMA 1 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x1C8–0x1FC	Reserved	—	—	—
0x200	MR2—DMA 2 mode register	R/W	All zeros	19.3.1.1/19-9

Table A-15. DMA Controller 1 Registers (continued)

DMA Controller 1—Block Base Address 0x2_1000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x204	SR2—DMA 2 status register	Mixed	All zeros	19.3.1.2/19-12
0x208	ECLNDAR2—DMA 2 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x20C	CLNDAR2—DMA 2 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x210	SATR2—DMA 2 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x214	SAR2—DMA 2 source address register	R/W	All zeros	19.3.1.5/19-16
0x218	DATR2—DMA 2 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x21C	DAR2—DMA 2 destination address register	R/W	All zeros	19.3.1.7/19-18
0x220	BCR2—DMA 2 byte count register	R/W	All zeros	19.3.1.8/19-19
0x224	ENLNDAR2—DMA 2 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x228	NLNDAR2—DMA 2 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19
0x230	ECLSDAR2—DMA 2 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x234	CLSDAR2—DMA 2 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x238	ENLSDAR2—DMA 2 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x23C	NLSDAR2—DMA 2 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22
0x240	SSR2—DMA 2 source stride register	R/W	All zeros	19.3.1.12/19-23
0x244	DSR2—DMA 2 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x248–0x27C	Reserved	—	—	—
0x280	MR3—DMA 3 mode register	R/W	All zeros	19.3.1.1/19-9
0x284	SR3—DMA 3 status register	Mixed	All zeros	19.3.1.2/19-12
0x288	ECLNDAR3—DMA 3 current link descriptor extended address register	R/W	All zeros	19.3.1.3/19-13
0x28C	CLNDAR3—DMA 3 current link descriptor address register	R/W	All zeros	19.3.1.3/19-13
0x290	SATR3—DMA 3 source attributes register	R/W	All zeros	19.3.1.4/19-15
0x294	SAR3—DMA 3 source address register	R/W	All zeros	19.3.1.5/19-16
0x298	DATR3—DMA 3 destination attributes register	R/W	All zeros	19.3.1.6/19-17
0x29C	DAR3—DMA 3 destination address register	R/W	All zeros	19.3.1.7/19-18
0x2A0	BCR3—DMA 3 byte count register	R/W	All zeros	19.3.1.8/19-19
0x2A4	ENLNDAR3—DMA 3 next link descriptor extended address register	R/W	All zeros	19.3.1.9/19-19
0x2A8	NLNDAR3—DMA 3 next link descriptor address register	R/W	All zeros	19.3.1.9/19-19

Table A-15. DMA Controller 1 Registers (continued)

DMA Controller 1—Block Base Address 0x2_1000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x2B0	ECLSDAR3—DMA 3 current list descriptor extended address register	R/W	All zeros	19.3.1.10/19-20
0x2B4	CLSDAR3—DMA 3 current list descriptor address register	R/W	All zeros	19.3.1.10/19-20
0x2B8	ENLSDAR3—DMA 3 next list descriptor extended address register	R/W	All zeros	19.3.1.11/19-22
0x2BC	NLSDAR3—DMA 3 next list descriptor address register	R/W	All zeros	19.3.1.11/19-22
0x2C0	SSR3—DMA 3 source stride register	R/W	All zeros	19.3.1.12/19-23
0x2C4	DSR3—DMA 3 destination stride register	R/W	All zeros	19.3.1.13/19-23
0x2C8–0x2FC	Reserved	—	—	—
0x300	DGSR—DMA general status register	R	All zeros	19.3.1.14/19-24

A.1.16 Display Interface Unit

Table A-16. Display Interface Unit Registers

Display Interface Unit—Block Base Address 0x2_C000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	DESC_1 — Pointer to the Area Descriptor of Plane1	R/W	All zeros	10.3.1.1/10-4
0x004	DESC_2 — Pointer to the Area Descriptor of Plane2	R/W	All zeros	10.3.1.2/10-5
0x008	DESC_3 — Pointer to the Area Descriptor of Plane3	R/W	All zeros	10.3.1.3/10-5
0x00c	GAMMA — Pointer to Gamma Table	R/W	All zeros	10.3.1.4/10-6
0x010	PALETTE — Pointer to Palette	R/W	All zeros	10.3.1.5/10-6
0x014	CURSOR — Pointer to Cursor Bitmap	R/W	All zeros	10.3.1.6/10-7
0x018	CURS_POS — Position of the cursor in the display	R/W	All zeros	10.3.1.7/10-7
0x01c	DIU_MODE — DIU Mode of Operation	R/W	All zeros	10.3.1.8/10-8
0x020	BGND — Background Color	R/W	All zeros	10.3.1.9/10-8
0x024	BGND_WB — Background Color in write back Mode	R/W	All zeros	10.3.1.10/10-9
0x028	DISP_SIZE — Display Size	R/W	All zeros	10.3.1.11/10-10
0x02c	WB_SIZE — Write back Plane Size	R/W	All zeros	10.3.1.12/10-10
0x030	WB_MEM_ADDR — Address to Store the write back Plane	R/W	All zeros	10.3.1.13/10-11
0x034	HSYN_PARA — Horizontal synchronization pulse parameters	R/W	All zeros	10.3.1.14/10-11
0x038	VSYN_PARA — Vertical synchronization pulse parameters	R/W	All zeros	10.3.1.15/10-12
0x03c	SYN_POL — Synchronization Signals Polarity	R/W	All zeros	10.3.1.16/10-13
0x040	THRESHOLDS — The Thresholds	R/W	0x0000_7800	10.3.1.17/10-13

Table A-16. Display Interface Unit Registers (continued)

Display Interface Unit—Block Base Address 0x2_C000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x044	INT_STATUS — Interrupt Status Register	R	All zeros	10.3.1.18/10-14
0x048	INT_MASK — Interrupt Mask Register	R/W	All zeros	10.3.1.19/10-15
0x04c	COLORBAR1 — Color #1 in the Color Bar, Black	R/W	0xFF00_0000	10.3.1.20/10-15
0x050	COLORBAR2 — Color #2 in the Color Bar, Blue	R/W	0xFF00_00FF	10.3.1.20/10-15
0x054	COLORBAR3 — Color #3 in the Color Bar, Cyan	R/W	0xFF00_7F7F	10.3.1.20/10-15
0x058	COLORBAR4 — Color #4 in the Color Bar, Green	R/W	0xFF00_FF00	10.3.1.20/10-15
0x05c	COLORBAR5 — Color #5 in the Color Bar, Yellow	R/W	0xFF7F_7F00	10.3.1.20/10-15
0x060	COLORBAR6 — Color #6 in the Color Bar, Red	R/W	0xFFFF_0000	10.3.1.20/10-15
0x064	COLORBAR7 — Color #7in the Color Bar, Purple	R/W	0xFF7F_007F	10.3.1.20/10-15
0x068	COLORBAR8 — Color #8 in the Color Bar, White	R/W	0xFFFF_FFFF	10.3.1.20/10-15
0x06c	FILLING — Input, output buffer filling status, for debug purpose	R	All zeros	10.3.1.21/10-17
0x070	PLUT — Priority Look Up Table	R/W	All zeros	10.3.1.22/10-18

A.1.17 Infrared Interface Controller 1

Table A-17. Infrared Interface Controller 1 (SIRI1/FIRI1) Registers

SIRI1—Block Base Address 0x2_D000 FIRI1—Block Base Address 0x2_D100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	SIRI receiver register (SRXD)	R	All zeros	14.4.1.1/14-5
0x040	SIRI transmitter register (STXD)	W	All zeros	14.4.1.2/14-6
0x080	SIRI control register 1 (SCR1)	R/W	All zeros	14.4.1.3/14-7
0x084	SIRI control register 2 (SCR2)	R/W	0x0000_0001	14.4.1.4/14-9
0x088	SIRI control register 3 (SCR3)	R/W	0x0000_0700	14.4.1.5/14-11
0x08C	SIRI control register 4 (SCR4)	R/W	0x0000_8000	14.4.1.6/14-12
0x090	SIRI FIFO control register (SFCR)	R/W	0x0000_0801	14.4.1.7/14-14
0x094	SIRI status register 1 (SSR1)	Mixed	0x0000_2040	14.4.1.8/14-15
0x098	SIRI status register 2 (SSR2)	Mixed	0x0000_4448	14.4.1.9/14-17
0x09C	SIRI escape character register (SESC)	R/W	0x0000_002B	14.4.1.10/14-18
0x0A0	SIRI escape timer register (STIM)	R/W	All zeros	14.4.1.11/14-19
0x0A4	SIRI BRM incremental register (SBIR)	R/W	All zeros	14.4.1.12/14-19
0x0A8	SIRI BRM modulator register (SBMR)	R/W	All zeros	14.4.1.13/14-20
0x0AC	SIRI baud rate count register (SBRC)	R	0x0000_0004	14.4.1.14/14-20

Table A-17. Infrared Interface Controller 1 (SIRI1/FIRI1) Registers (continued)

SIRI1—Block Base Address 0x2_D000 FIRI1—Block Base Address 0x2_D100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0B0	SIRI one millisecond register (SONEMS)	R/W	All zeros	14.4.1.15/14-21
0x0B4	SIRI test register (STS)	R/W	0x0000_0060	14.4.1.16/14-21
0x100	FIRI transmit control register (FIRITCR)	Mixed	All zeros	14.4.1.17/14-23
0x104	FIRI transmit count register (FIRITCTR)	R/W	All zeros	14.4.1.18/14-25
0x108	FIRI receive control register (FIRIRCR)	R/W	All zeros	14.4.1.19/14-26
0x10C	FIRI transmit status register (FIRITSR)	Mixed	All zeros	14.4.1.20/14-28
0x110	FIRI receive status register (FIRIRSR)	Mixed	All zeros	14.4.1.21/14-29
0x114	Transmitter FIFO	W	0x00	
0x118	Receiver FIFO	R/W	0x00	
0x11C	FIRI control register (FIRICR)	R/W	All zeros	14.4.1.22/14-30

A.1.18 Infrared Interface Controller 2

Table A-18. Infrared Interface Controller 2 (SIRI2/FIRI2) Registers

SIRI2—Block Base Address 0x2_E000 FIRI2—Block Base Address 0x2_E100				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000–0xFFC	Infrared Interface Controller 2 (SIRI2/FIRI2) All registers defined for SIRI1/FIRI1 above are also defined for SIRI2/FIRI2; the offsets are the same, but they have different block base addresses.			14.4/14-4

A.2 Programmable Interrupt Controller (PIC)

The programmable interrupt controller (PIC) follows the OpenPIC programming model which requires a larger register address space than the 4 Kbytes allocated to other blocks within the general utilities space. For this reason, the PIC is allocated the second 256 Kbytes of CCSR space (0x4_0000–0x6_FFFF).

A.2.1 PIC—Global Registers

Table A-19. PIC Global Registers

PIC Global Registers—Block Base Address 0x4_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0000	BRR1—Block revision register 1	R	0x0040_0300	11.3.1.1/11-18
0x0010	BRR2—Block revision register 2	R	0x0000_0001	11.3.1.2/11-19
0x0020–0x0030	Reserved	—	—	—

Table A-19. PIC Global Registers (continued)

PIC Global Registers—Block Base Address 0x4_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0040	IPIDR0—Interprocessor interrupt 0 (IPI 0) dispatch register	W	All zeros	11.3.8.1/11-49
0x0050	IPIDR1—IPI 1 dispatch register			
0x0060	IPIDR2—IPI 2 dispatch register			
0x0070	IPIDR3—IPI 3 dispatch register			
0x0080	CTPR—Current task priority register	R/W	0x0000_000F	11.3.8.2/11-49
0x0090	WHOAMI—Who am I register	R	n/a	11.3.8.3/11-50
0x00A0	IACK—Interrupt acknowledge register	R	All zeros	11.3.8.4/11-51
0x00B0	EOI—End of interrupt register	W	All zeros	11.3.8.5/11-51
0x00C0–0x0FF0	Reserved	—	—	—
0x1000	FRR—Feature reporting register	R	0x0067_0002	11.3.1.3/11-19
0x1010	Reserved	—	—	—
0x1020	GCR—Global configuration register	R/W	All zeros	11.3.1.4/11-20
0x1030	Reserved	—	—	—
0x1040–0x1070	Vendor reserved	—	—	—
0x1080	VIR—Vendor identification register	R	All zeros	11.3.1.5/11-21
0x1090	PIR—Processor core initialization register	R/W	All zeros	11.3.1.6/11-21
0x1098	PRR—Processor reset register	R/W	All zeros	11.3.1.7/11-22
0x10A0	IPIVPR0—IPI 0 vector/priority register	R/W	0x8000_0000	11.3.1.8/11-22
0x10B0	IPIVPR1—IPI 1 vector/priority register			
0x10C0	IPIVPR2—IPI 2 vector/priority register			
0x10D0	IPIVPR3—IPI 3 vector/priority register			
0x10E0	SVR—Spurious vector register	R/W	0x0000_FFFF	11.3.1.9/11-23
Global Timer Group A Registers				
0x10F0	TFRRA—Timer frequency reporting register (Group A)	R/W	All zeros	11.3.2.1/11-24
0x1100	GTCCRA0—Global timer 0 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x1110	GTBCRA0—Global timer 0 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25
0x1120	GTVPRA0—Global timer 0 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x1130	GTDRA0—Global timer 0 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x1140	GTCCRA1—Global timer 1 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x1150	GTBCRA1—Global timer 1 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25
0x1160	GTVPRA1—Global timer 1 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x1170	GTDRA1—Global timer 1 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x1180	GTCCRA2—Global timer 2 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x1190	GTBCRA2—Global timer 2 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25

Table A-19. PIC Global Registers (continued)

PIC Global Registers—Block Base Address 0x4_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x11A0	GTVPRA2—Global timer 2 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x11B0	GTDRA2—Global timer 2 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x11C0	GTCCRA3—Global timer 3 current count register (Group A)	R	All zeros	11.3.2.2/11-25
0x11D0	GTBCRA3—Global timer 3 base count register (Group A)	R/W	0x8000_0000	11.3.2.3/11-25
0x11E0	GTVPRA3—Global timer 3 vector/priority register (Group A)	R/W	0x8000_0000	11.3.2.4/11-26
0x11F0	GTDRA3—Global timer 3 destination register (Group A)	R/W	0x0000_0001	11.3.2.5/11-27
0x1200–0x12F0	Reserved	—	—	—
0x1300	TCRA—Timer control register (Group A)	R/W	All zeros	11.3.2.6/11-27
0x1308	ERQSR—External interrupt summary register	R	All zeros	11.3.3.1/11-29
0x1310	IRQSR0—IRQ_OUT summary register 0	R	All zeros	11.3.3.2/11-30
0x1320	IRQSR1—IRQ_OUT summary register 1	R	All zeros	11.3.3.3/11-31
0x1324	IRQSR2—IRQ_OUT summary register 2	R	All zeros	11.3.3.4/11-31
0x1330	CISR0—Critical interrupt summary register 0	R	All zeros	11.3.3.5/11-32
0x1340	CISR1—Critical interrupt summary register 1	R	All zeros	11.3.3.6/11-33
0x1344	CISR2—Critical interrupt summary register 2	R	All zeros	11.3.3.7/11-33
0x1350	PM0MR0—Performance monitor 0 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34
0x1360	PM0MR1—Performance monitor 0 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1364	PM0MR2—Performance monitor 0 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1370	PM1MR0—Performance monitor 1 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34
0x1380	PM1MR1—Performance monitor 1 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1384	PM1MR2—Performance monitor 1 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x1390	PM2MR0—Performance monitor 2 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34
0x13A0	PM2MR1—Performance monitor 2 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13A4	PM2MR2—Performance monitor 2 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13B0	PM3MR0—Performance monitor 3 mask register 0	R/W	0xFFFF_FFFF	11.3.4.1/11-34
0x13C0	PM3MR1—Performance monitor 3 mask register 1	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13C4	PM3MR2—Performance monitor 3 mask register 2	R/W	0xFFFF_FFFF	11.3.4.2/11-34
0x13D0–0x13F0	Reserved	—	—	—
0x1400	MSGR0—Message register 0	R/W	All zeros	11.3.5.1/11-36
0x1410	MSGR1—Message register 1	R/W	All zeros	11.3.5.1/11-36
0x1420	MSGR2—Message register 2	R/W	All zeros	11.3.5.1/11-36
0x1430	MSGR3—Message register 3	R/W	All zeros	11.3.5.1/11-36
0x1440–0x14F0	Reserved	—	—	—
0x1500	MER—Message enable register	R/W	All zeros	11.3.5.2/11-36

Table A-19. PIC Global Registers (continued)

PIC Global Registers—Block Base Address 0x4_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x1510	MSR—Message status register	R/W	All zeros	11.3.5.3/11-37
0x1520–0x15F0	Reserved	—	—	—
0x1600	MSIR0—Shared message signaled interrupt register 0	RC	All zeros	11.3.6.1/11-37
0x1610	MSIR1—Shared message signaled interrupt register 1	RC	All zeros	11.3.6.1/11-37
0x1620	MSIR2—Shared message signaled interrupt register 2	RC	All zeros	11.3.6.1/11-37
0x1630	MSIR3—Shared message signaled interrupt register 3	RC	All zeros	11.3.6.1/11-37
0x1640	MSIR4—Shared message signaled interrupt register 4	RC	All zeros	11.3.6.1/11-37
0x1650	MSIR5—Shared message signaled interrupt register 5	RC	All zeros	11.3.6.1/11-37
0x1660	MSIR6—Shared message signaled interrupt register 6	RC	All zeros	11.3.6.1/11-37
0x1670	MSIR7—Shared message signaled interrupt register 7	RC	All zeros	11.3.6.1/11-37
0x1680–0x1700	Reserved	—	—	—
0x1720	MSISR—Shared message signaled interrupt status register	R	All zeros	11.3.6.2/11-38
0x1740	MSIIR—Shared message signaled interrupt index register	W	All zeros	11.3.6.3/11-38
0x1750–0x20E0	Reserved	—	—	—
Global Timer Group B Registers				
0x20F0	TFRRB—Timer frequency reporting register group B	R/W	All zeros	11.3.2.1/11-24
0x2100	GTCCRB0—Global timer current count register group B 0	R	All zeros	11.3.2.2/11-25
0x2110	GTBCRB0—Global timer base count register group B 0	R/W	0x8000_0000	11.3.2.3/11-25
0x2120	GTVPRB0—Global timer vector/priority register group B 0	R/W	0x8000_0000	11.3.2.4/11-26
0x2130	GTDRB0—Global timer destination register group B 0	R/W	0x0000_0001	11.3.2.5/11-27
0x2140	GTCCRB1—Global timer current count register group B 1	R	All zeros	11.3.2.2/11-25
0x2150	GTBCRB1—Global timer base count register group B 1	R/W	0x8000_0000	11.3.2.3/11-25
0x2160	GTVPRB1—Global timer vector/priority register group B 1	R/W	0x8000_0000	11.3.2.4/11-26
0x2170	GTDRB1—Global timer destination register group B 1	R/W	0x0000_0001	11.3.2.5/11-27
0x2180	GTCCRB2—Global timer current count register group B 2	R	All zeros	11.3.2.2/11-25
0x2190	GTBCRB2—Global timer base count register group B 2	R/W	0x8000_0000	11.3.2.3/11-25
0x21A0	GTVPRB2—Global timer vector/priority register group B 2	R/W	0x8000_0000	11.3.2.4/11-26
0x21B0	GTDRB2—Global timer destination register group B 2	R/W	0x0000_0001	11.3.2.5/11-27
0x21C0	GTCCRB3—Global timer current count register group B 3	R	All zeros	11.3.2.2/11-25
0x21D0	GTBCRB3—Global timer base count register group B 3	R/W	0x8000_0000	11.3.2.3/11-25
0x21E0	GTVPRB3—Global timer vector/priority register group B 3	R/W	0x8000_0000	11.3.2.4/11-26
0x21F0	GTDRB3—Global timer destination register group B 3	R/W	0x0000_0001	11.3.2.5/11-27
0x2200–0x22F0	Reserved	—	—	—

Table A-19. PIC Global Registers (continued)

PIC Global Registers—Block Base Address 0x4_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x2300	TCRB—Timer control register (Group B)	R/W	All zeros	11.3.2.6/11-27
0x2310–0xFFF0	Reserved	—	—	—

A.2.2 PIC—Interrupt Source Registers

Table A-20. PIC Interrupt Source Registers

PIC Interrupt Source Registers—Block Base Address 0x5_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0000	EIVPR0—External interrupt 0 (IRQ0) vector/priority register or PEX1-INTA vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0010	EIDR0—External interrupt 0 (IRQ0) destination register or PEX1-INTA destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0020	EIVPR1—External interrupt 1 (IRQ1) vector/priority register or PEX1-INTB vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0030	EIDR1—External interrupt 1 (IRQ1) destination register or PEX1-INTB destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0040	EIVPR2—External interrupt 2 (IRQ2) vector/priority register or PEX1-INTC vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0050	EIDR2—External interrupt 2 (IRQ2) destination register or PEX1-INTC destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0060	EIVPR3—External interrupt 3 (IRQ3) vector/priority register or PEX1-INTD vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0070	EIDR3—External interrupt 3 (IRQ3) destination register or PEX1-INTD destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0080	EIVPR4—External interrupt 4 (IRQ4) vector/priority register or PEX2-INTA vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0090	EIDR4—External interrupt 4 (IRQ4) destination register or PEX2-INTA destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x00A0	EIVPR5—External interrupt 5 (IRQ5) vector/priority register or PEX2-INTB vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x00B0	EIDR5—External interrupt 5 (IRQ5) destination register or PEX2-INTB destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x00C0	EIVPR6—External interrupt 6 (IRQ6) vector/priority register or PEX2-INTC vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x00D0	EIDR6—External interrupt 6 (IRQ6) destination register or PEX2-INTC destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x00E0	EIVPR7—External interrupt 7 (IRQ7) vector/priority register or PEX2-INTD vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x00F0	EIDR7—External interrupt 7 (IRQ7) destination register or PEX2-INTD destination register	R/W	0x0000_0001	11.3.7.2/11-43

Table A-20. PIC Interrupt Source Registers (continued)

PIC Interrupt Source Registers—Block Base Address 0x5_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0100	EIVPR8—External interrupt 8 (IRQ8) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0110	EIDR8—External interrupt 8 (IRQ8) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0120	EIVPR9—External interrupt 9 (IRQ9) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0130	EIDR9—External interrupt 9 (IRQ9) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0140	EIVPR10—External interrupt 10 (IRQ10) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0150	EIDR10—External interrupt 10 (IRQ10) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0160	EIVPR11—External interrupt 11 (IRQ11) vector/priority register	R/W	0x8000_0000	11.3.7.1/11-42
0x0170	EIDR11—External interrupt 11 (IRQ11) destination register	R/W	0x0000_0001	11.3.7.2/11-43
0x0180–0x01F0	Reserved	—	—	—
0x0200	IIVPR0—Internal interrupt 0 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0210	IIDR0—Internal interrupt 0 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0220	IIVPR1—Internal interrupt 1 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0230	IIDR1—Internal interrupt 1 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0240	IIVPR2—Internal interrupt 2 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0250	IIDR2—Internal interrupt 2 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0260	IIVPR3—Internal interrupt 3 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0270	IIDR3—Internal interrupt 3 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0280	IIVPR4—Internal interrupt 4 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0290	IIDR4—Internal interrupt 4 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x02A0	IIVPR5—Internal interrupt 5 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x02B0	IIDR5—Internal interrupt 5 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x02C0	IIVPR6—Internal interrupt 6 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x02D0	IIDR6—Internal interrupt 6 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x02E0	IIVPR7—Internal interrupt 7 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x02F0	IIDR7—Internal interrupt 7 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0300	IIVPR8—Internal interrupt 8 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0310	IIDR8—Internal interrupt 8 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0320	IIVPR9—Internal interrupt 9 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0330	IIDR9—Internal interrupt 9 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0340	IIVPR10—Internal interrupt 10 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0350	IIDR10—Internal interrupt 10 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0360	IIVPR11—Internal interrupt 11 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0370	IIDR11—Internal interrupt 11 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0380	IIVPR12—Internal interrupt 12 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44

Table A-20. PIC Interrupt Source Registers (continued)

PIC Interrupt Source Registers—Block Base Address 0x5_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0390	IIDR12—Internal interrupt 12 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x03A0	IIVPR13—Internal interrupt 13 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x03B0	IIDR13—Internal interrupt 13 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x03C0	IIVPR14—Internal interrupt 14 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x03D0	IIDR14—Internal interrupt 14 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x03E0	IIVPR15—Internal interrupt 15 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x03F0	IIDR15—Internal interrupt 15 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0400	IIVPR16—Internal interrupt 16 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0410	IIDR16—Internal interrupt 16 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0420	IIVPR17—Internal interrupt 17 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0430	IIDR17—Internal interrupt 17 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0440	IIVPR18—Internal interrupt 18 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0450	IIDR18—Internal interrupt 18 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0460	IIVPR19—Internal interrupt 19 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0470	IIDR19—Internal interrupt 19 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0480	IIVPR20—Internal interrupt 20 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0490	IIDR20—Internal interrupt 20 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x04A0	IIVPR21—Internal interrupt 21 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x04B0	IIDR21—Internal interrupt 21 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x04C0	IIVPR22—Internal interrupt 22 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x04D0	IIDR22—Internal interrupt 22 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x04E0	IIVPR23—Internal interrupt 23 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x04F0	IIDR23—Internal interrupt 23 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0500	IIVPR24—Internal interrupt 24 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0510	IIDR24—Internal interrupt 24 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0520	IIVPR25—Internal interrupt 25 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0530	IIDR25—Internal interrupt 25 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0540	IIVPR26—Internal interrupt 26 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0550	IIDR26—Internal interrupt 26 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0560	IIVPR27—Internal interrupt 27 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0570	IIDR27—Internal interrupt 27 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0580	IIVPR28—Internal interrupt 28 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0590	IIDR28—Internal interrupt 28 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x05A0	IIVPR29—Internal interrupt 29 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x05B0	IIDR29—Internal interrupt 29 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x05C0	IIVPR30—Internal interrupt 30 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44

Table A-20. PIC Interrupt Source Registers (continued)

PIC Interrupt Source Registers—Block Base Address 0x5_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x05D0	IIDR30—Internal interrupt 30 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x05E0	IIVPR31—Internal interrupt 31 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x05F0	IIDR31—Internal interrupt 31 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0600	IIVPR32—Internal interrupt 32 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0610	IIDR32—Internal interrupt 32 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0620	IIVPR33—Internal interrupt 33 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0630	IIDR33—Internal interrupt 33 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0640	IIVPR34—Internal interrupt 34 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0650	IIDR34—Internal interrupt 34 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0660	IIVPR35—Internal interrupt 35 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0670	IIDR35—Internal interrupt 35 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0680	IIVPR36—Internal interrupt 36 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0690	IIDR36—Internal interrupt 36 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x06A0	IIVPR37—Internal interrupt 37 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x06B0	IIDR37—Internal interrupt 37 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x06C0	IIVPR38—Internal interrupt 38 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x06D0	IIDR38—Internal interrupt 38 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x06E0	IIVPR39—Internal interrupt 39 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x06F0	IIDR39—Internal interrupt 39 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0700	IIVPR40—Internal interrupt 40 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0710	IIDR40—Internal interrupt 40 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0720	IIVPR41—Internal interrupt 41 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0730	IIDR41—Internal interrupt 41 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0740	IIVPR42—Internal interrupt 42 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0750	IIDR42—Internal interrupt 42 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0760	IIVPR43—Internal interrupt 43 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0770	IIDR43—Internal interrupt 43 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0780	IIVPR44—Internal interrupt 44 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0790	IIDR44—Internal interrupt 44 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x07A0	IIVPR45—Internal interrupt 45 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x07B0	IIDR45—Internal interrupt 45 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x07C0	IIVPR46—Internal interrupt 46 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x07D0	IIDR46—Internal interrupt 46 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x07E0	IIVPR47—Internal interrupt 47 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x07F0	IIDR47—Internal interrupt 47 destination register	R/W	0x0000_0001	11.3.7.4/11-45

Table A-20. PIC Interrupt Source Registers (continued)

PIC Interrupt Source Registers—Block Base Address 0x5_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0800–0x15F0	Reserved	—	—	—
0x0800	IIVPR48—Internal interrupt 48 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0810	IIDR48—Internal interrupt 48 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0820	IIVPR49—Internal interrupt 49 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0830	IIDR49—Internal interrupt 49 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0840	IIVPR50—Internal interrupt 50 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0850	IIDR50—Internal interrupt 50 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0860	IIVPR51—Internal interrupt 51 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0870	IIDR51—Internal interrupt 51 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0880	IIVPR52—Internal interrupt 52 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0890	IIDR52—Internal interrupt 52 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x08A0	IIVPR53—Internal interrupt 53 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x08B0	IIDR53—Internal interrupt 53 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x08C0	IIVPR54—Internal interrupt 54 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x08D0	IIDR54—Internal interrupt 54 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x08E0	IIVPR55—Internal interrupt 55 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x08F0	IIDR55—Internal interrupt 55 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0900	IIVPR56—Internal interrupt 56 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0910	IIDR56—Internal interrupt 56 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0920	IIVPR57—Internal interrupt 57 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0930	IIDR57—Internal interrupt 57 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0940	IIVPR58—Internal interrupt 58 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0950	IIDR58—Internal interrupt 58 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0960	IIVPR59—Internal interrupt 59 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0970	IIDR59—Internal interrupt 59 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x0980	IIVPR60—Internal interrupt 60 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x0990	IIDR60—Internal interrupt 60 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x09A0	IIVPR61—Internal interrupt 61 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x09B0	IIDR61—Internal interrupt 61 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x09C0	IIVPR62—Internal interrupt 62 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x09D0	IIDR62—Internal interrupt 62 destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x09E0	IIVPR63—Internal interrupt 63 vector/priority register	R/W	0x8080_0000	11.3.7.3/11-44
0x09F0	IIDR63—Internal interrupt 63 destination register	R/W	0x0000_0001	11.3.7.3/11-44
0x1600	MIVPR0—Messaging interrupt 0 (MSG 0) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46

Table A-20. PIC Interrupt Source Registers (continued)

PIC Interrupt Source Registers—Block Base Address 0x5_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x1610	MIDR0—Messaging interrupt 0 (MSG 0) destination register	R/W	0x0000_0001	11.3.7.6/11-46
0x1620	MIVPR1—Messaging interrupt 1 (MSG 1) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46
0x1630	MIDR1—Messaging interrupt 1 (MSG 1) destination register	R/W	0x0000_0001	11.3.7.6/11-46
0x1640	MIVPR2—Messaging interrupt 2 (MSG 2) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46
0x1650	MIDR2—Messaging interrupt 2 (MSG 2) destination register	R/W	0x0000_0001	11.3.7.6/11-46
0x1660	MIVPR3—Messaging interrupt 3 (MSG 3) vector/priority register	R/W	0x8000_0000	11.3.7.5/11-46
0x1670	MIDR3—Messaging interrupt 3 (MSG 3) destination register	R/W	0x0000_0001	11.3.7.4/11-45
0x1680–0x1BF0	Reserved	—	—	—
0x1C00	MSIVPR0—Shared message signaled interrupt vector/priority register 0	R/W	0x8000_0000	11.3.6.4/11-39
0x1C10	MSIDR0—Shared message signaled interrupt destination register 0	R/W	0x0000_0001	11.3.6.5/11-40
0x1C20	MSIVPR1—Shared message signaled interrupt vector/priority register 1	R/W	0x8000_0000	11.3.6.4/11-39
0x1C30	MSIDR1—Shared message signaled interrupt destination register 1	R/W	0x0000_0001	11.3.6.5/11-40
0x1C40	MSIVPR2—Shared message signaled interrupt vector/priority register 2	R/W	0x8000_0000	11.3.6.4/11-39
0x1C50	MSIDR2—Shared message signaled interrupt destination register 2	R/W	0x0000_0001	11.3.6.5/11-40
0x1C60	MSIVPR3—Shared message signaled interrupt vector/priority register 3	R/W	0x8000_0000	11.3.6.4/11-39
0x1C70	MSDIR3—Shared message signaled interrupt destination register 3	R/W	0x0000_0001	11.3.6.5/11-40
0x1C80	MSIVPR4—Shared message signaled interrupt vector/priority register 4	R/W	0x8000_0000	11.3.6.4/11-39
0x1C90	MSIDR4—Shared message signaled interrupt destination register 4	R/W	0x0000_0001	11.3.6.5/11-40
0x1CA0	MSIVPR5—Shared message signaled interrupt vector/priority register 5	R/W	0x8000_0000	11.3.6.4/11-39
0x1CB0	MSIDR5—Shared message signaled interrupt destination register 5	R/W	0x0000_0001	11.3.6.5/11-40
0x1CC0	MSIVPR6—Shared message signaled interrupt vector/priority register 6	R/W	0x8000_0000	11.3.6.4/11-39
0x1CD0	MSIDR6—Shared message signaled interrupt destination register 6	R/W	0x0000_0001	11.3.6.5/11-40
0x1CE0	MSIVPR7—Shared message signaled interrupt vector/priority register 7	R/W	0x8000_0000	11.3.6.4/11-39

Table A-20. PIC Interrupt Source Registers (continued)

PIC Interrupt Source Registers—Block Base Address 0x5_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x1CF0	MSDIR7—Shared message signaled interrupt destination register 7	R/W	0x0000_0001	11.3.6.5/11-40
0x1D00–0xFFF0	Reserved	—	—	—

A.2.3 PIC—Processor (per-CPU) Registers

Table A-21. PIC Processor (per-CPU) Registers

PIC Processor (per-CPU) Registers—Block Base Address 0x6_0000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x0000–0x0030	Reserved	—	—	—
0x0040	IPIDR0—Processor core 0 interprocessor 0 dispatch register	W	all zeros	11.3.8.1/11-49
0x0050	IPIDR1—Processor core 0 interprocessor 1 dispatch register			
0x0060	IPIDR2—Processor core 0 interprocessor 2 dispatch register			
0x0070	IPIDR3—Processor core 0 interprocessor 3 dispatch register			
0x0080	CTPR0—Processor core 0 current task priority register	R/W	0x0000_000F	11.3.8.2/11-49
0x0090	WHOAMI0—Processor core 0 who am I register	R	n/a	11.3.8.3/11-50
0x00A0	IACK0—Processor core 0 interrupt acknowledge register	R	all zeros	11.3.8.4/11-51
0x00B0	EOI0—Processor core 0 end of interrupt register	W	all zeros	11.3.8.5/11-51
0x00C0–0x0FF0	Reserved	—	—	—
0x1000–0x1030	Reserved	—	—	—
0x1040	IPIDR0—Processor core 1 interprocessor 0 dispatch register	W	all zeros	11.3.8.1/11-49
0x1050	IPIDR1—Processor core 1 interprocessor 1 dispatch register			
0x1060	IPIDR2—Processor core 1 interprocessor 2 dispatch register			
0x1070	IPIDR3—Processor core 1 interprocessor 3 dispatch register			
0x1080	CTPR1—Processor core 1 current task priority register	R/W	0x0000_000F	11.3.8.2/11-49
0x1090	WHOAMI1—Processor core 1 who am I register	R	n/a	11.3.8.3/11-50
0x10A0	IACK1—Processor core 1 interrupt acknowledge register	R	all zeros	11.3.8.4/11-51
0x10B0	EOI1—Processor core 1 end of interrupt register	W	all zeros	11.3.8.5/11-51
0x10C0–0xFFFC	Reserved	—	—	—

A.3 Device-Specific Utilities

The device-specific utilities registers control functions that are not particular to a functional unit but to the device as a whole; they occupy the highest 256 Kbytes of CCSR space (0xE_0000–0xF_FFFF).

A.3.1 Global Utilities

Table A-22. Global Utilities Registers

Global Utilities—Block Base Address 0xE_0000				
Offset	**Register**	**Access**	**Reset** [1]	**Section/Page**
Power-On Reset Configuration Values				
0x000	PORPLLSR—POR PLL ratio status register	R	0x00*nn*_*n*0*nn*	23.4.1.1/23-5
0x004	PORBMSR—POR boot mode status register	R	0x*nnnn*_0000	23.4.1.2/23-6
0x008	PORIMPCR—POR I/O impedance control register	Mixed	0x000*n*_007F	23.4.1.3/23-8
0x00C	PORDEVSR—POR I/O device status register	R	see ref***.	23.4.1.4/23-9
0x010	PORDBGMSR—POR debug mode status register	R	see ref.	23.4.1.5/23-10
0x018	PORGEN—POR general usage register	R	0x*nn*0*n*_0000	23.4.1.6/23-10
0x020	PORCIR—POR configuration information register	R	see ref.	23.4.1.7/23-11
Signal Multiplexing and GPIO Controls				
0x030	GPIOCR—GPIO control register	R/W	All zeros	23.4.1.8/23-12
0x060	PMUXCR—Alternate function signal multiplex control	R/W	All zeros	23.4.1.9/23-14
Device Disables				
0x070	DEVDISR—Device disable control	R/W	All zeros	23.4.1.10/23-16
0x074	DEVDISR2—Device disable control 2	R/W	All zeros	23.4.1.10/23-16
Power Management Registers				
0x080	POWMGTCSR—Power management status and control register	Mixed	All zeros	23.4.1.12/23-19
Interrupt and Reset Status and Control Registers				
0x090	MCPSUMR—Machine check summary register	w1c	All zeros	23.4.1.13/23-20
0x094	RSTRSCR—Reset request status and control register	R	All zeros	23.4.1.14/23-21
Version Registers				
0x0A0	PVR—Processor version register	R	0x8004_*nnnn*	23.4.1.15/23-21
0x0A4	SVR—System version register	R	0x80A0_00*nn*	23.4.1.16/23-22
Reset Registers				
0x0B0	RSTCR—Reset control register	R/W	All zeros	23.4.1.17/23-22

Table A-22. Global Utilities Registers (continued)

Global Utilities—Block Base Address 0xE_0000				
Offset	**Register**	**Access**	**Reset** [1]	**Section/Page**
0x0C0	eLBCVSELCR—eLBC voltage select control register	R/W	All zeros	23.4.1.18/23-23
Clock Control				
0x800	CKDVDR—Clock divide register	R/W	All zeros	23.4.1.19/23-23
Configuration and Status				
0x900	IRCR—Infrared control register	R/W	All zeros	23.4.1.20/23-25
0x904	MCMCR—MCM control register	R/W	All zeros	23.4.1.21/23-25
0x908	DMACR—DMA control register	R/W	All zeros	23.4.1.22/23-26
0x914	eLBCCR—eLBC control register	R/W	All zeros	23.4.1.23/23-27
DDR Control				
0xB28	DDRCLKDR—DDR clock disable register	R/W	All zeros	23.4.1.24/23-28
Debug, Clock, and SerDes Control				
0xE00	CLKOCR—CLK_OUT control register	R/W	All zeros	23.4.1.25/23-29
0xF04	SRDS1DCR0—SerDes1 (×4) debug control register 0	R/W	0x1105_4418	23.4.1.26/23-30
0xF08	SRDS1DCR1—SerDes2 (×4) debug control register 1	R/W	0x0000_000*n*	23.4.1.27/23-31
0xF40	SRDS2DCR0—SerDes2 (×8) debug control register 0	R/W	0x1105_4418	23.4.1.28/23-31
0xF44	SRDS2DCR1—SerDes2 (×8) debug control register 1	R/W	0x0000_000*n*	23.4.1.29/23-32

[1] Bits indicated with *n* are set from configuration signals.

A.3.2 Performance Monitor

Table A-23. Performance Monitor Registers

Performance Monitor—Block Base Address 0xE_1000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000	PMGC0—Performance monitor global control register	R/W	All zeros	24.3.1.1/24-5
0x010	PMLCA0—Performance monitor local control register A0	R/W	All zeros	24.3.1.2/24-5
0x014	PMLCB0—Performance monitor local control register B0	R/W	All zeros	24.3.1.2/24-5
0x018	PMC0 (lower)—Performance monitor counter 0 lower	R/W	All zeros	24.3.2.1/24-9
0x01C	PMC0 (upper)—Performance monitor counter 0 upper	R/W	All zeros	24.3.2.1/24-9
0x020	PMLCA1—Performance monitor local control register A1	R/W	All zeros	24.3.1.2/24-5
0x024	PMLCB1—Performance monitor local control register B1	R/W	All zeros	24.3.1.2/24-5
0x028	PMC1—Performance monitor counter 1	R/W	All zeros	24.3.2.1/24-9

Table A-23. Performance Monitor Registers (continued)

Performance Monitor—Block Base Address 0xE_1000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x030	PMLCA2—Performance monitor local control register A2	R/W	All zeros	24.3.1.2/24-5
0x034	PMLCB2—Performance monitor local control register B 2	R/W	All zeros	24.3.1.2/24-5
0x038	PMC2—Performance monitor counter 2	R/W	All zeros	24.3.2.1/24-9
0x040	PMLCA3—Performance monitor local control register A3	R/W	All zeros	24.3.1.2/24-5
0x044	PMLCB3—Performance monitor local control register B3	R/W	All zeros	24.3.1.2/24-5
0x048	PMC3—Performance monitor counter 3	R/W	All zeros	24.3.2.1/24-9
0x050	PMLCA4—Performance monitor local control register A4	R/W	All zeros	24.3.1.2/24-5
0x054	PMLCB4—Performance monitor local control register B4	R/W	All zeros	24.3.1.2/24-5
0x058	PMC4—Performance monitor counter 4	R/W	All zeros	24.3.2.1/24-9
0x060	PMLCA5—Performance monitor local control register A5	R/W	All zeros	24.3.1.2/24-5
0x064	PMLCB5—Performance monitor local control register B 5	R/W	All zeros	24.3.1.2/24-5
0x068	PMC5—Performance monitor counter 5	R/W	All zeros	24.3.2.1/24-9
0x070	PMLCA6—Performance monitor local control register A6	R/W	All zeros	24.3.2.1/24-9
0x074	PMLCB6—Performance monitor local control register B6	R/W	All zeros	24.3.1.2/24-5
0x078	PMC6—Performance monitor counter 6	R/W	All zeros	24.3.2.1/24-9
0x080	PMLCA7—Performance monitor local control register A7	R/W	All zeros	24.3.1.2/24-5
0x084	PMLCB7—Performance monitor local control register B7	R/W	All zeros	24.3.1.2/24-5
0x088	PMC7—Performance monitor counter 7	R/W	All zeros	24.3.2.1/24-9
0x090	PMLCA8—Performance monitor local control register A8	R/W	All zeros	24.3.1.2/24-5
0x094	PMLCB8—Performance monitor local control register B8	R/W	All zeros	24.3.1.2/24-5
0x098	PMC8—Performance monitor counter 8	R/W	All zeros	24.3.2.1/24-9
0x0A0	PMLCA9—Performance monitor local control register A9	R/W	All zeros	24.3.1.2/24-5
0x0A4	PMLCB9—Performance monitor local control register B9	R/W	All zeros	24.3.1.2/24-5
0x0A8	PMC9—Performance monitor counter 9	R/W	All zeros	24.3.2.1/24-9

A.3.3 Watchpoint Monitor and Trace Buffer

Table A-24. Watchpoint Monitor and Trace Buffer Registers

Watchpoint Monitor and Trace Buffer—Block Base Address 0xE_2000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
Watchpoint Monitor Registers				
0x000	WMCR0—Watchpoint monitor control register 0	R/W	All zeros	25.3.1.1/25-8

Table A-24. Watchpoint Monitor and Trace Buffer Registers (continued)

Watchpoint Monitor and Trace Buffer—Block Base Address 0xE_2000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x004	WMCR1—Watchpoint monitor control register 1	R/W	All zeros	25.3.1.2/25-10
0x008	WMAHR—Watchpoint monitor address high register	R/W	All zeros	25.3.1.3/25-11
0x00C	WMAR—Watchpoint monitor address register	R/W	All zeros	25.3.1.4/25-11
0x010	WMAMHR—Watchpoint monitor address mask high register	R/W	All zeros	25.3.1.5/25-12
0x014	WMAMR—Watchpoint monitor address mask register	R/W	All zeros	25.3.1.6/25-12
0x018	WMTMR—Watchpoint monitor transaction mask register	R/W	All zeros	25.3.1.7/25-12
0x01C	WMSR—Watchpoint monitor status register	R/W	All zeros	25.3.1.8/25-14
Trace Buffer Registers				
0x040	TBCR0—Trace buffer control register 0	R/W	All zeros	25.3.2.1/25-15
0x044	TBCR1—Trace buffer control register 1	R/W	All zeros	25.3.2.2/25-17
0x048	TBAHR—Trace buffer address high register	R/W	All zeros	25.3.2.3/25-17
0x04C	TBAR—Trace buffer address register	R/W	All zeros	25.3.2.4/25-18
0x050	TBAMHR—Trace buffer address mask high register	R/W	All zeros	25.3.2.5/25-18
0x054	TBAMR—Trace buffer address mask register	R/W	All zeros	25.3.2.6/25-19
0x058	TBTMR—Trace buffer transaction mask register	R/W	All zeros	25.3.2.7/25-19
0x05C	TBSR—Trace buffer status register	R/W	All zeros	25.3.2.8/25-20
0x060	TBACR—Trace buffer access control register	R/W	All zeros	25.3.2.9/25-20
0x064	TBADHR—Trace buffer access data high register	R/W	All zeros	25.3.2.10/25-21
0x068	TBADR—Trace buffer access data register	R/W	All zeros	25.3.2.11/25-21
Context ID Registers				
0x0A0	PCIDR—Programmed context ID register	R/W	All zeros	25.3.3.1/25-22
0x0A4	CCIDR—Current context ID register	R/W	All zeros	25.3.3.2/25-23
Other Registers				
0x0B0	TOSR—Trigger output source register	R/W	All zeros	25.3.4.1/25-23

A.3.4 Watchdog Timer

Table A-25. Watchdog Timer Registers

Watchdog Timer—Block Base Address 0xE_4000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x000–0x003	Reserved	—	—	—
0x004	System watchdog control register (SWCRR)	R/W	0x0000_0007	18.2.1/18-3

Table A-25. Watchdog Timer Registers (continued)

Watchdog Timer—Block Base Address 0xE_4000				
Offset	**Register**	**Access**	**Reset**	**Section/Page**
0x008	System watchdog count register (SWCNR)	R	0x0000_FFFF	18.2.2/18-4
0x00C–0x00D	Reserved	—	—	—
0x00E	System watchdog service register (SWSRR)	R/W	All zeros	18.2.3/18-5

Appendix B
Revision History

This appendix provides a list of the major differences between revisions of the *MPC8610 Integrated Host Processor Reference Manual*.

This appendix provides a list of the major differences between the *MPC8610 Integrated Host Processor Reference Manual*, Revision 0 through Revision 1.

B.1 Changes From Revision 0 to Revision 1

2.4/2-11	Added Section 2.4, “Configuration, Control, and Status Registers.”
4.4.4.2/4-21	Added Section 4.4.4.2.1, “Minimum Frequency Requirements.”
Chapter 5	Removed references to Deep Sleep mode, Dynamic Frequency Switching (DFS), and MPX address and data bus parity.
5.2/5-5	Changed bullet from: “Supports big- and little-endian modes...” to say “Supports both big-endian and PPC_LE modes...”
Chapter 6	Removed references to DFS.
6.1.4.2/6-9	Updated cross-reference to Section 23.4.1.16, “System Version Register (SVR).”
8.3.2.1/8-6	In Table 8-3, “Memory Interface Signals—Detailed Signal Descriptions,” updated signal description of MA[15:0].:
8.4.1.7/8-20	In Table 8-12, “DDR_SDRAM_CFG Field Descriptions,” updated the 8_BE (bit 13) field description note.
8.4.1.19/8-32	Added a note.
8.5/8-43	Updated the second sentence in the third paragraph as follows: “Bank sizes up to 4 Gbits are supported, providing up to a maximum of 16 Gbits of DDR main memory per chip select.”
8.5.2/8-50	Changed the firstsentence of the third paragraph.
8.5.6/8-63	Added note after first paragraph.
8.5.11/8-69	Added the following text to the end of the section: “In 32-bit mode, Table 8-56 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.”
9.1.2/9-2	Added bullet at the end of list of features.

9.3.1.1/9-10	In Figure 9-2, ”Base Registers (BRn),” changed reset value of reserved bit 30 to 0.
9.3.1.12/9-30	Added note that LTEATR and LTEAR may not capture accurate information for errors that occur when an FCM special operation is in progress.
9.3.1.14/9-32	in Table 9-21, “LBCR Field Descriptions,” updated AHD (bit 10) field description.
9.4/9-40	Revised note at the end of the section.
9.4.1.2/9-42	Updated second paragraph. In addition, added two paragraphs to the end of the section after Figure 9-29.
9.4.1.7/9-45	Added note to end of section describing consequences of long FCM transactions and GPCM/UPM transactions.
9.4.2/9-46	In Figure 9-31, “GPCM Basic Read Timing (XACS = 0, ACS = 1x, TRLX = 0),” updated address numbering A.
9.4.2.4/9-56	Updated Figure 9-41, “External Termination of GPCM Access (PLL Bypass Mode).”
9.4.3.4.1/9-72	In Table 9-36, “Boot Bank Field Values after Reset for FCM as Boot Controller,” updated reset value for SCY.
9.4.4/9-74	Removed Section 9.4.4.5, “Synchronous Sampling of LUPWAIT for Early Transfer Acknowledge.”
9.4.4.4.1/9-81	In Table 9-38, “RAM Word Field Descriptions,” added note to UTA (bit 29) and LAST (bit 31) field descriptions.
9.4.4.4.7/9-86	Modified the last two sentences of the first paragraph. In addition, added note regarding bank addresses on muliple-bank DRAM and SDRAM devices.
9.4.4.4.9/9-88	Added a second paragraph to the section.
9.5.1/9-90	Removed the following text from introductory paragraph: “The local bus can be used either with separate address and data buses or with a multiplexed address/data bus.”
9.5.1.4/9-92	Modified the last sentence of second paragraph.
9.5.4/9-95	Changed “FMR[OP] = 01” to “FMR[OP] = 11” in the first paragraph of several subsections.
Chapter 11	Changed “Message Shared,” to “Message Signaled,” throughout.
11.1.1/11-2	Removed the following sub-bullet: “Multi-cast delivery mode for interprocessor and global timer interrupts allowing these interrupts to be routed to either core 0 or 1, or both cores”
11.1.3.2/11-5	Added the following sentence to the end of the section: It is required that in pass-through mode the internal and external interrupt targets should be *int*, which is by default.”
11.3.3.1/11-29	In the NOTE, changed “state” to “logic level.” In addition, in Table 11-21, “ERQSR Field Descriptions,” updated the values in EINT*n* (bits 0–11) field description.

11.3.7.2/11-43	In Figure 11-40, “External Interrupt Destination Registers (EIDRs),” made bit P0 (bit 31) read-only, removed P1 (bit 30), and changed access to “Mixed.” In addition, modified the introductory text for the register.
11.3.7.6/11-46	In Figure 11-44, “Messaging Interrupt Destination Registers (MIDRn),” and Table 11-44, “MIDRn Field Descriptions,” removed bits EP, CI0, and CI1.
12.5.4.3/12-17	Removed bullets. In addition, exchanged the Master and Slave columns in Table 12-15, “Module Behavior at Startup During a Data Transfer.”
15.2.3.3/15-5	Updated the paragraph directly following Figure 15-13.
15.4.1.2/15-11	In Table 15-5, “SPIE Field Descriptions,” modified LT (bit 17) field description.
16.3.1/16-25	Removed Section 16.3.1.14, “SSI Test Register (STR),” and Section 16.3.1.15, “SSI Option Register (SOR).”
19.4.1.1.4/19-28	Revised first sentence of Step 1.
19.4.2/19-33	Added note that a single DMA transfer in any of the direct or chaining modes must not cross a 16-Gbyte (34-bit) address boundary.
21.3.2.3/21-11	In Table 21-6, “PEX_OTB_CPL_TOR Field Descriptions,” revised TC (bits 8–31) field description.
21.3.2.5/21-12	In Figure 21-6, “PCI Express Configuration Register (PEX_CONFIG),” and Table 21-8, “PEX_CONFIG Field Descriptions,” added SB_EN (bit 17) to support southbridge mode.
21.3.6/21-30	Revised descriptions for PEX_ERR_CAP_R0, PEX_ERR_CAP_R1, PEX_ERR_CAP_R2, and PEX_ERR_CAP_R3.
21.3.6.4/21-36	In Table 21-26, “PCI Express Error Capture Status Register Field Descriptions,” updated encodings for GSID (bits 26–30) field description.
21.3.7/21-44	Added statement that the PCI Express controller internal CSR registers are not accessible by inbound PCI Express configuration transactions.
21.3.7.1.1/21-44	In last bullet, added the statement that, “In this case [no bus number matches], PEX_ERR_DR[ICCA] is set.”
21.3.9.11/21-75	In Table 21-86, “PCI Express Link Control Register Field Description,” updated RL (bit 5) field description.
21.3.10/21-83	In Figure 21-102, “PCI Express Extended Configuration Space,” revised range for Internal CSR space in to “0x400–0x6FF.”
21.4.1.8/21-104	Revised first paragraph of the section.
21.4.1.9/21-104	In Table 21-122, “PCI Express ATMU Outbound Messages,” updated the first eight rows (Assert_INTA through Deassert_INTD) to be supported.
21.4.1.10/21-108	In Table 21-126, “Error Conditions,” revised entry for Outbound response/Internal platform response with error to describe behavior when poison data occurs in the middle of the packet.
22.3/22-2	Updated all register figures in this section.

24.4.7/24-14	In Table 24-11, “Event Assignments: MCM, DDR Controller, eLBC,” removed MCM “dispatch to” events.
25.3.2.1/25-15	In Table 25-15, “TBCR0 Field Descriptions,” updated the note in ECEN (bit 3) and NECEN (bit 4) field descriptions.
25.4.1/25-24	In Table 25-29, “Source and Target ID Encodings,” updated encoding 1000 to be CCSR (target only).
25.4.5.1/25-27	In Table 25-30, “CMD Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 000),” changed “010101” to “10101” in CMDSID (bits 5–9) and CMDTID (bits 10–13) field descriptions.
25.4.5.1/25-27	In Table 25-32, “PCI Express Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 100 or 011),” updated PEXADDR (bits 34–63) field description to say the following: PCI Express Address bits 31–2.”
25.4.5.1/25-27	In Table 25-33, “PCI Trace Buffer Entry Field Descriptions (TBCR1[IFSEL] = 010),” changed the size of the last bit field to 28–63, and renamed it PCIADDR.