460EX/EXr/GT

PPC460EX/EXr/GT Embedded Processor

User's Manual

PPC460EX/EXr/GT Embedded Processor User's Manual

Special Note: The PPC440H6 processor core information is in a separate document titled the PPC440H6 and PPC464 Processor User's Manual.

Printed in the United States of America, March 30, 2012.

Applied Micro Circuits Corporation
215 Moffett Park Drive, Sunnyvale, CA 94089
Phone: (408) 542-8600 — (800) 840-6055 — Fax: (408) 542-8601
http://www.apm.com

Contents

Figures

Tables

About This Book

This user's manual provides the architectural overview, programming model, and detailed information about the registers, the instruction set, and operations of the AppliedMicro PowerPC™ 460EX/EXr/GT (PPC460EX/EXr/GT) processor. This is a 32-bit reduced instruction set computer (RISC) processor.

The PPC460EX/EXr/GT RISC processor features:

- PowerPC Architecture
- Single-cycle execution for most instructions
- Instruction cache unit and data cache unit
- Support for little endian operation
- Interrupt interface for one critical and one non-critical interrupt signal
- JTAG interface

Who Should Use This Book

This book is for system hardware and software developers, and for application developers who need to understand the PPC460EX/EXr/GT. The reader should already understand network processor design, network system design, operating systems, RISC processing, and design for testability.

How to Use This Book

This book describes the PPC460EX/EXr/GT device architecture, programming model, external interfaces, internal registers, and instruction set. It contains the following chapters, arranged in parts:

- Part I Introduction
  - *Overview* on page 95
  - *On-Chip Buses* on page 101
- Part II Processor
  - *Programming Model* on page 153
  - *Cache Operations* on page 183
  - *Memory Management* on page 185
  - *L2 Cache/SRAM Controller* on page 187
  - *SRAM Controller* on page 203
  - *Bootstrap Operations* on page 213
  - *Clocking* on page 237
- Part III System Operations
  - *Interrupt Controller Operations* on page 255
  - *Interrupt Handling* on page 267
  - *Floating Point Unit Interrupts and Exceptions* on page 269
  - *Reset and Initialization* on page 283
  - *Timer Facilities* on page 305
  - *Debugging* on page 307
  - *Clock and Power Management* on page 309
- Part IV External Interfaces
  - *Security Function* on page 315
  - *PCI Controller (PPC460EX-N and PPC460GT-N Only)* on page 409
  - *PCI Express (PCIE)* on page 481

To help readers find material in these chapters, the book contains:

Conventions

The following is a list of notational conventions frequently used in this manual.

$\overline{\text{ActiveLow}}$	An overbar indicates an active-low signal.
n or n	A decimal number
0x*n*	A hexadecimal number
0b*n*	A binary number
=	Assignment
$\wedge$	AND logical operator
$\neg$	NOT logical operator
$\vee$	OR logical operator
$\oplus$	Exclusive-OR (XOR) logical operator
+	Twos complement addition
–	Twos complement subtraction, unary minus
$\times$	Multiplication
$\div$	Division yielding a quotient
%	Remainder of an integer division; (33 % 32) = 1.
$\|\|$	Concatenation
$=, \neq$	Equal, not equal relations
<, >	Signed comparison relations
$\underline{u}, \underline{u}$ <, >	Unsigned comparison relations
if...then...else...	Conditional execution; if *condition* then *a* else *b*, where *a* and *b* represent one or more pseudocode statements. Indenting indicates the ranges of *a* and *b*. If *b* is null, the else does not appear.
do	Do loop. "to" and "by" clauses specify incrementing an iteration variable; "while" and "until" clauses specify terminating conditions. Indenting indicates the scope of a loop.
leave	Leave innermost do loop or do loop specified in a leave statement.
FLD	An instruction or register field
FLD_b	A bit in a named instruction or register field
$FLD_{b:b}$	A range of bits in a named instruction or register field
$FLD_{b,b,\ldots}$	A list of bits, by number or name, in a named instruction or register field
REG_b	A bit in a named register
$REG_{b:b}$	A range of bits in a named register
$REG_{b,b,\ldots}$	A list of bits, by number or name, in a named register
REG[FLD]	A field in a named register
REG[FLD, FLD . . .]	A list of fields in a named register

REG[FLD:FLD]	A range of fields in a named register
GPR(r)	General Purpose Register (GPR) r, where $0 \leq r \leq 31$.
(GPR(r))	The contents of GPR r, where $0 \leq r \leq 31$.
DCR(DCRN)	A Device Control Register (DCR) specified by the DCRF field in an **mfdcr** or **mtdcr** instruction
SPR(SPRN)	An SPR specified by the SPRF field in an **mfspr** or **mtspr** instruction
TBR(TBRN)	A Time Base Register (TBR) specified by the TBRF field in an **mftb** instruction
GPRs	RA, RB, . . .
(Rx)	The contents of a GPR, where x is A, B, S, or T
(RA|0)	The contents of the register RA or 0, if the RA field is 0.
CR_{FLD}	The field in the condition register pointed to by a field of an instruction.
$c_{0:3}$	A 4-bit object used to store condition results in compare instructions.
^{n}b	The bit or bit value b is replicated n times.
xx	Bit positions which are don't-cares.
CEIL(x)	Least integer $\geq x$.
EXTS(x)	The result of extending x on the left with sign bits.
PC	Program counter.
RESERVE	Reserve bit; indicates whether a process has reserved a block of storage.
CIA	Current instruction address; the 32-bit address of the instruction being described by a sequence of pseudocode. This address is used to set the next instruction address (NIA). Does not correspond to any architected register.
NIA	Next instruction address; the 32-bit address of the next instruction to be executed. In pseudocode, a successful branch is indicated by assigning a value to NIA. For instructions that do not branch, the NIA is CIA +4.
MS(addr, n)	The number of bytes represented by n at the location in main storage represented by addr.
EA	Effective address; the 32-bit address, derived by applying indexing or indirect addressing rules to the specified operand, that specifies a location in main storage.
EA_b	A bit in an effective address.
$EA_{b:b}$	A range of bits in an effective address.
ROTL((RS),n)	Rotate left; the contents of RS are shifted left the number of bits specified by n.
MASK(MB,ME)	Mask having 1s in positions MB through ME (wrapping if MB > ME) and 0s elsewhere.
instruction(EA)	An instruction operating on a data or instruction cache block associated with an EA.

Part I. Introduction

1. Overview

The AppliedMicro PowerPC 460EX/EXr/GT, 32-bit RISC embedded processor, referred to as the PPC460EX/EXr/GT, is a system-on-a-chip (SOC) design that integrates the PowerPC 440H6 processor core with a rich set of peripheral controllers.

- PPC460EX/EXr/GT is a 32-bit implementation of Book-E Enhanced PowerPC Architecture.
- Multiple PPC460EX/EXr/GT interfaces that provide a peripheral mix for a wide variety of applications.
- Debug support for a wide range of hardware and software development tools.
- System power management, low power dissipation, and small form factor
- RoHS compliant (lead-free)

Because the PPC460EX, PPC460EXr, and PPC460GT are closely related designs, they are identified separately only where differences exist. Throughout this user's manual, PPC460EX/EXr/GT is used where text is common to all.

Notes:

1. Refer to the “PowerPC 440H6 and PPC464 Processor User's Manual” for information on the 440H6 processor core, programming model and the descriptions of the caches, MMU, debug features, processor timer facility (watchdog/fixed interval timer/programmable interval timer), special purpose registers (SPR) and interrupts.
2. Block diagrams, detail feature lists, part numbers and PVR numbers are provided in the PowerPC 460EX, 460EXr, and 460GT Embedded Processor Data Sheets.

1.1 PPC460EX, PPC460EXr, and PPC460GT Differences

Differences between the processors are limited to the PCI, USB, SATA, Ethernet, RapidIO, and the processor version register number (PVR).

PPC460EX/EXr features:
- SATA (multiplexed with 1-lane PCI Express)
- No RapidIO
- One 1-lane and One 4-lane PCI Express port
- USB 2.0 Host
- USB 2.0 OTG
- Up to two 10/100/1000 Ethernet ports
- PCI (PPC460EX-N only)
- No PCI (PPC460EXr and PPC460EX-T)

PPC460GT features:
- No SATA
- RapidIO (multiplexed with 4-lane PCI Express)
- One 1-lane and One 4-lane PCI Express port
- No USB 2.0 Host
- No USB 2.0 OTG
- Up to four 10/100/1000 Ethernet ports
- PCI (PPC460GT-N only)
- No PCI (PPC460GT-T only)

1.2 PPC460EX/EXr/GT Embedded Processor Features

The PPC460EX/EXr/GT provides high performance and low power consumption with a CPU processor executing at sustained speeds approaching one cycle per instruction. On-chip instruction and data caches reduce chip count and design complexity in systems and improve system throughput. The CPU, combined with a diverse peripheral mix, provides an ideal foundation for systems incorporating system-on-a-chip (SOC) designs such as the PPC460EX/EXr/GT. This section provides a list of features that are implemented in the PPC460EX/EXr/GT.

1.2.1 PowerPC 460EX/EXr/GT Processor Features

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the CPU in this embedded processor.

The PPC460EX, PPC460EXr, and the PPC460GT all contain the PPC440H6 processor core. The PPC440H6 processor core is described in the PPC440H6 and PPC464 Processor User's Manual.

1.2.2 Floating Point Unit (FPU)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the FPU in this embedded processor.

1.2.3 Internal Buses

The PPC460EX and PPC460EXr contain four internal buses: the processor local bus (PLB), the advanced high-performance bus (AHB), the on-chip peripheral bus (OPB), and the device control register (DCR) bus. The PPC460GT does not have an AHB.

High performance devices such as the processor, the DDR SDRAM memory controller, PCI Express, the Ethernet MAL, and DMA utilize the PLB. Lower bandwidth I/O interfaces such as communications and timer interfaces utilize the OPB. The daisy-chained DCR bus provides a lower bandwidth path for passing status and control information between the processor and the other on-chip peripheral functions.

1.2.3.1 Processor Local Bus (PLB)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the PLB in this embedded processor.

1.2.3.2 On-chip Peripheral Bus (OPB)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the OPB in this embedded processor.

1.2.3.3 Device Control Register (DCR) Bus

The daisy-chained DCR bus provides a path for passing status and control information between the processor and the other on-chip functions. All DCRs are 32 bits in width with 10-bit addressing.

1.2.3.4 Advanced High-Performance Bus (AHB) (PPC460EX and PPC460EXr only)

Please refer to the PPC460EX or PPC460EXr Data Sheet for a list of features provided by the AHB in this embedded processor.

1.2.4 L2 Cache/SRAM

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the L2 Cache/SRAM in this embedded processor.

1.2.5 On-Chip Memory (OCM)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the OCM in this embedded processor.

1.2.6 External Bus Controller (EBC)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the EBC in this embedded processor.

1.2.7 NAND Flash Controller

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the NAND Flash Controller in this embedded processor.

1.2.8 I2O/DMA Controller

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the I2O/DMA Controller in this embedded processor.

1.2.9 DMA Controller

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the DMA Controller in this embedded processor.

1.2.10 USB 2.0 Interface (PPC460EX and PPC460EXr only)

Please refer to the PPC460EX or PPC460EXr Data Sheet for a list of features provided by the USB interface in this embedded processor.

1.2.11 DDR2/1 SDRAM Controller

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the DDR2/1 SDRAM Controller in this embedded processor.

1.2.12 PCI Controller (PPC460EX-N and PPC460GT-N only)

Please refer to the PPC460EX-N or PPC460GT-N Data Sheet for a list of features provided by PCI in this embedded processor.

1.2.13 PCI Express

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by PCI Express in this embedded processor.

1.2.14 Security Function

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the Security Function in this embedded processor.

1.2.15 Serial ATA Interface (SATA) (PPC460EX and PPC460EXr only)

Please refer to the PPC460EX or PPC460EXr Data Sheet for a list of features provided by the SATA interface in this embedded processor.

1.2.16 UART

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the UART in this embedded processor.

1.2.17 Serial RapidIO (SRIO) (PPC460GT only)

Please refer to the PPC460GT Data Sheet for a list of features provided by the RapidIO in this embedded processor.

1.2.18 IIC Bus Interface

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the IIC Bus interface in this embedded processor.

1.2.19 Serial Communication Port Interface (SCP/SPI)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the SCP/SPI interface in this embedded processor.

1.2.20 General Purpose I/O (GPIO) Controller

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the GPIO controller in this embedded processor.

1.2.21 Universal Interrupt Controller (UIC)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the UIC in this embedded processor.

1.2.22 Gigabit Ethernet

The Ethernet support for the PPC460EX and PPC460EXr provides two Gigabit (10/100/1000 Mbps) interfaces (GMII/RGMII).

The Ethernet support for the PPC460GT provides up to four Gigabit (10/100/1000 Mbps) interfaces (RGMII).

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the Ethernet interface in this embedded processor.

1.2.23 General Purpose Timer (GPT)

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the GPT controller in this embedded processor.

1.2.24 JTAG

Please refer to the PPC460EX, PPC460EXr, or PPC460GT Data Sheet for a list of features provided by the JTAG interface in this embedded processor.

2. On-Chip Buses

The on-chip bus structure, which consists of the processor local bus (PLB), on-chip peripheral bus (OPB), advanced high-performance bus (AHB)(PPC460EX/EXr only), and device control register (DCR) bus, provides a connection among all the masters and slaves on the chip buses. The block diagram in the *Power PC 460EX, 460EXr, or 460GT Embedded Processor Data Sheet* illustrates the on-chip bus structure of each of these three chips.

PLB4 is a high performance bus used to access memory through bus interface units. The PLB master and slave assignments for the PPC460EX/EXr/GT are listed in *PLB Master and Slave Assignments* on page 102. See *Processor Local Bus* on page 101.

Lower performance peripherals (such as serial ports, the Ethernet controller, the external bus controller, general purpose I/O, and IIC controllers) are attached to the OPB. A bridge between the PLB and OPB enables data transfers between PLB masters and OPB slaves. See *On-Chip Peripheral Bus* on page 130.

Higher performance peripherals (such as the USB 2.0 Host controller and OTG interfaces, and SATA) are attached to the AHB (PPC460EX/EXr only). A bridge between the PLB and AHB enables data transfers between the masters and slaves of both buses. See *Advanced High Performance Bus (PPC460EX/EXr only)* on page 133.

The DCR bus is used primarily to access status and control registers of the various PLB and OPB masters and slaves. The DCR bus off-loads status and control read and write transfers from the PLB. See *Device Control Register (DCR) Bus* on page 150.

2.1 Processor Local Bus

The processor local bus is a high-performance on-chip bus. The PLB supports read and write data transfers between master and slave devices equipped with a PLB interface and connected through PLB signals.

Each PLB master is attached to the PLB through separate address, read data and write data buses, and transfer qualifier signals. PLB slaves are attached to the PLB through shared, but decoupled, address, read data and write data buses, and transfer control and status signals for each data bus.

Access to the PLB is granted through a central arbitration mechanism that enables masters to compete for bus ownership. This arbitration mechanism provides for fixed and fair priority schemes.

Timing for all PLB signals is provided by a clock source that is shared by all PLB masters and slaves.

2.1.1 PLB Features

- Overlapping of read and write transfers allows two concurrent data transfers for maximum bus utilization
- Decoupled address and data buses support split-bus transaction capability for improved bandwidth
- Address pipelining reduces overall bus latency by allowing the latency associated with a new request to be overlapped with an ongoing data transfer in the same direction
- Late master request abort capability reduces latency associated with aborted requests
- Four levels of request priority for each master allow PLB implementations with various arbitration schemes
- Byte-enable capability allows for unaligned transfers and odd-byte transfers.
- Support for fixed length burst transfers
- Guarded and unguarded memory transfers allow a slave device to enable or disable the prefetching of instructions or data

- DMA buffered, peripheral to memory, memory to peripheral, and DMA memory to memory operations are supported

2.1.2 PLB Master and Slave Assignments

Table 2-1 lists the PLB4 masters and slaves provided in the PPC460EX/EXr/GT.

Table 2-1. PPC460EX/EXr/GT PLB4 Master and Slave Assignments

PLB Agent	PLB Masters and Slaves	Master/Slave No.
Processor instruction Cache Unit (ICU) read or L2	Master	0
Processor Data Cache Unit (DCU) read or L2	Master	1
PCI controller (PCI0)	Master Slave HB	2 1
PCI Express 1 controller (PCIE1)/Serial RapidIO (SRIO)	Master Slave HB	3 2
Processor Data Cache Unit (DCU) write or L2	Master	4
PCI Express 0 controller (PCIE0)	Master Slave HB	5 3
DMA to PLB controller	Master	6
I2O/DMA (HSDMA)	Master Slave LL	7 1
I2O/DMA (I2O/FIFO)	Master	8
Security Function (EIP94)	Master Slave LL	9 4
Memory Access Layer (MAL)	Master	10
PLB to AHB bridge (PLB4XAHB)	Master Slave LL	11 6
Memory Queue (MQ)	Slave LL Slave HB	0 0
Internal SRAM controller using L2 cache	Slave LL	2
PLB-to-OPB bridge	Slave LL	3
PCI Express interrupt handler	Slave HB	4
TRNG and PKA (EIP-PKP)	Slave LL	5
Internal SRAM (64KB OCM)	Slave LL	7
Note: HB is the high-bandwidth PLB1 segment. LL is the low-latency PLB0 segment.		

2.1.3 PLB Master Priority Assignment

Each PLB master can be programmed to use one of four priority levels during PLB transfers, enabling the system designer to tune PLB transfer priorities to the requirements of a particular application. For example, if an application always requires DMA to SDRAM transfers to have the lowest latency, the DMA master can be programmed to the highest PLB master priority. This causes the PLB arbiter to grant DMA access requests before granting the access requests of any other master.

A register associated with each master controls the priority of that master. *Table 2-2* lists the PLB4 masters and the register fields controlling the priority of the masters. Priorities range from 0b00 (lowest) to 0b11 (highest).

Table 2-2. Registers Controlling PLB Master Priority Assignments

Master ID	Description	Register Field	Comments
0	Processor Instruction Cache Unit (ICU)	SDR0_CP440[IRF] SDR0_CP440[IRT] SDR0_CP440[IRS]	IcuRdFetch priority setting IcuRdTouch priority setting IcuRdSpec priority setting
1	Processor Data Cache Unit (DCU) read	SDR0_CP440[DRU] SDR0_CP440[DRT] SDR0_CP440[DRNC] SDR0_CP440[DRLC]	DcuRdUrgent priority setting DcuRdTouch priority setting DcuRdNonCache priority DcuRdLdCache priority setting
2	PCI (legacy) bus controller	PCI0_BRDGOPT1[LPRQ]	Identified as PCI0
3	PCI Express 1 bus controller	PEGPL1_CFG[MPRI]	Identified as PE1
4	Processor Data Cache Unit (DCU) write	SDR0_CP440[DWF] SDR0_CP440[DWS] SDR0_CP440[DWU]	DcuWrFlush priority setting DcuWrStore priority setting DcuWrUrgent priority setting
5	PCI Express 0 bus controller	PEGPL0_CFG[MPRI]	Identified as PE0
6	DMA (DMA2P40)	DMA2P40_CRN[CP]	Identified as DMA2P40
7	I2O/DMA (HSDMA)	I2O_DMA_CFG[DXEPR] I2O_DMA_CFG[DFMPP]	HSDMA Data Transfer Engine HSDMA FIFO Manager
8	I2O/DMA (I2O/FIFO)	I2O_ICTL[PLBPR]	Identified as I2ODMA
9	Security function	CRYPTO_DMA_CFG[PRI]	Identified as EIP94
10	Media Access Layer (MAL)	MAL0_CFG[RPP] MAL0_CFG[WPP]	MAL Read MAL Write
11	PLB-to-AHB Bridge	PLB4XAHB_A2PCTL[PLBRP]	Identified as PLB4XAHB

Note: PLB master priority assignments are application-dependent, and must be considered carefully in order to prevent potential lockouts of lower priority masters. For most applications, assigning a priority of 0b10 to each master is a useful starting point. See *PLB4 Arbiter Control Register (PLB4An_ACR)* on page 118 for information about programming the PLB4A0_ACR to control PLB priority mode and priority order. The PLB0_ACR must be set to "fair" mode arbitration. This helps to prevent various lockout scenarios.

Table 2-3. PLB4 SDRs

Mnemonic	Register	DCR Address	Access	Page
SDR0_AMP0	PLB4 Alternate Master Priority 0	0x0240	R/W	104
SDR0_AMP1	PLB4 Alternate Master Priority 1	0x0241	R/W	106
SDR0_CP440	PPC440 CPU Control	0x0180	R/W	107
SDR0_MIRQ0	Master Interrupt Request 0	0x0260	R/W	108
SDR0_MIRQ1	Master Interrupt Request 1	0x0261	R/W	109
SDR0_MIRQ2	Master Interrupt Request 2	0x0262	R/W	109
SDR0_SLPIPE	PLB Slave Address Pipeline	0x0220	R/W	111

2.1.3.1 PLB4 Alternate Master Priority Register 0 (SDR0_AMP0)

This is a 32-bit read/write register containing alternative PLB4 master priority setting, which allow overriding the PLB priority specified by the SDR0_CP440, PCI0_BRGOPT1, PEGPL0_CFG, PEGPL1_CFG, DMA2P40_CR and I2O_DMA_CFG registers.

Reset value = 0 for all fields.

Figure 2-1. PLB4 Alternate Master Priority Register 0 (SDR0_AMP0)

0:1	AICURP	Alternate ICU Read Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by SDR0_CP440[IRF, IRT, IRS].
2:3	ADCURP	Alternate DCU Read Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by SDR0_CP440[DRU, DRT, DRNC, DRLC].
4:5	APCI0P	Alternate PCI0 Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by PCI0_BRDGOPT1[LPRQ].
6:7	APE1P	Alternate PCIE1 Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by PEGPL1_CFG[MPRI].
8:9	ADCUWP	Alternate DCU Write Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by SDR0_CP440[DWF, DWS, DWU].

10:11	APE0P	Alternate PCIE0 Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by PEGPL0_CFG[MPRI].
12:13	ADMA0P	Alternate DMA2P40 Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by DMA2P40_CRN[CP].
14:15	AHSDMAP	Alternate HSDMA Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by I2O_DMA_CFG[DXEPR] and I2O_DMA_CFG[DFMPP]
16	ICURP	ICU Read Priority 0 Controls own priority 1 Uses alternate priority	
17	DCURP	DCU Read Priority 0 Controls own priority 1 Uses alternate priority	
18	PCI0P	PCI0 Priority 0 Controls own priority 1 Uses alternate priority	
19	PE1P	PCIE1 Priority 0 Controls own priority 1 Uses alternate priority	
20	DCUWP	DCU Write Priority 0 Controls own priority 1 Uses alternate priority	
21	PE0P	PCIE0 Priority 0 Controls own priority 1 Uses alternate priority	
22	DMA2P40P	DMA2P40 Priority 0 Controls own priority 1 Uses alternate priority	
23	HSDMAP	HSDMA Priority 0 Controls own priority 1 Uses alternate priority	
24:31		Reserved	

2.1.3.2 PLB4 Alternate Master Priority Register 1 (SDR0_AMP1)

This is a 32-bit read/write register containing alternative PLB4 master priority setting, which allow overriding the PLB priority specified by the I2O_ICTL, CRYP0_DMA_CFG, MAL0_CFG and PLB4XAHB_A2PCTL registers.

Reset value = 0 for all fields.

Figure 2-2. Alternate PLB4 Master Priority Register 1 (SDR0_AMP1)

0:1	AI2OP	Alternate I2O/FIFO Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by I2O_ICTL[PLBPR]
2:3	AEIP94P	Alternate EIP94 Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by CRYPTO_DMA_CFG[PRI]
4:5	AMALP	Alternate MAL Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by MAL0_CFG[RPP] and MAL0_CFG[WPP]
6:7	AAHBP	Alternate PLB4XAHB Priority 00 Lowest 01 10 11 Highest	Overrides priority specified by PLB4XAHB_A2PCTL[PLBRP]
8:15		Reserved	
16	I2OP	I2O/FIFO Priority 0 Controls own priority 1 Uses alternate priority	
17	EIP94P	Security Priority 0 Controls own priority 1 Uses alternate priority	
18	MALP	MAL Priority 0 Controls own priority 1 Uses alternate priority	
19	AHBP	PLB4XAHB Priority 0 Controls own priority 1 Uses alternate priority	
20:31		Reserved	

2.1.3.3 CPU Control Register (SDR0_CP440)

This is a 32-bit read/write register that controls processor priorities

Reset value = see individual field definitions.

Figure 2-3. CPU Register (SDR0_CP440)

0:1		Reserved	
2:3	RL	ROM Location 00 EBC 01 PCI 10 NDFC 11 Reserved	Specifies the boot source. Reset value = SDR0_SDSTP1[RL]
4:5	DRU	DcuRdUrgent 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with an urgent state in which two or more read data cache operations are pending, waiting for the previous request to be serviced. PLB master priority is updated to this value when in urgent state regardless of instruction type. Reset value = 0b11.
6:7	DRT	DcuRdTouch 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with **dcbt** instructions except when in urgent state. Reset value = 0b10.
8:9	DRNC	DcuRdNonCache 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with non-cacheable load instructions except when in urgent state. Reset value = 0b10.
10:11	DRLC	DcuRdLdCache 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with cacheable load instructions except when in urgent state. Reset value = 0b10.
12:13	DWF	DcuWrFlush 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with flush instructions except when in urgent state. Reset value = 0b10.
14:15	DWS	DcuWrStore 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with store instructions except when in urgent state. Reset value = 0b10.
16:17	DWU	DcuWrUrgent 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with an urgent state in which two or more write data cache operations are pending, waiting for the previous request to be serviced. PLB master priority is updated to this value when in urgent state regardless of instruction type. Reset value = 0b11.
18:19	IRF	IcuRdFetch 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with non-speculative ICU accesses Reset value = 0b10.

20:21	IRT	IcuRdTouch 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with **icbt** instructions. Reset value = 0b10.
22:23	IRS	IcuRdSpec 00 Lowest 01 10 11 Highest	2-bit PLB priority level associated with speculative ICU accesses. Reset value = 0b10.
24:25	PW	Pulse Width 00 TBD% duty cycle 01 Hi/Low ratio of TBD%/TBD% 10 Hi/Low ratio of TBD%/TBD% 11 Hi/Low ratio of TBD%/TBD%	Select optimum duty cycle of OSC within the CPU. Reset value = 0b00.
26:31		Reserved	

2.1.3.4 PLB4 Master Interrupt Request Register 0 (SDR0_MIRQ0)

This is a 32-bit read/write register that controls interrupt requests from PLB masters 0 to 3.

Reset value = 0 for all fields.

Figure 2-4. Master Interrupt Request Register 0 (SDR0_MIRQ0)

0:1		Reserved	
2:7	ICURIRQ	ICU Read Interrupt Request Bit: 2 Interrupt request from DDR SDRAM LL 3 Interrupt request from PLB-to-OPB bridge 4 Interrupt request from PLB-to-AHB bridge 5 Interrupt request from DDR SDRAM HB 6 Interrupt request from PE1 or SRIO 7 Interrupt request from PE0	Master 0
8:9		Reserved	
10:15	DCURIRQ	DCU Read Interrupt Request Bit: 10 Interrupt request from DDR SDRAM LL 11 Interrupt request from PLB-to-OPB bridge 12 Interrupt request from PLB-to-AHB bridge 13 Interrupt request from DDR SDRAM HB 14 Interrupt request from PE1 or SRIO 15 Interrupt request from PE0	Master 1
16:17		Reserved	
18:23	PCI0IRQ	PCI0 Interrupt Request Bit: 18 PCI0 interrupt request from DDR SDRAM LL 19 PCI0 interrupt request from PLB-to-OPB bridge 20 PCI0 interrupt request from PLB-to-AHB bridge 21 PCI0 interrupt request from DDR SDRAM HB 22 PCI0 interrupt request from PE1 or SRIO 23 PCI0 interrupt request from PE0	Master 2
24:25		Reserved	

26:31	PE1IRQ	PE1 Interrupt Request Bit: 26 Interrupt request from DDR SDRAM LL 27 Interrupt request from PLB-to-OPB bridge 28 Interrupt request from PLB-to-AHB bridge 29 Interrupt request from DDR SDRAM HB 30 Interrupt request from PE1 or SRIO 31 Interrupt request from PE0	Master 3

2.1.3.5 PLB4 Master Interrupt Request Register 1 (SDR0_MIRQ1)

This is a 32-bit read/write register that controls interrupt requests from PLB masters 4 to 7.

Reset value = 0 for all fields.

Figure 2-5. Master Interrupt Request Register 1 (SDR0_MIRQ1)

0:1		Reserved	
2:7	DCUWIRQ	DCU Write Interrupt Request Bit: 2 Interrupt request from DDR SDRAM LL 3 Interrupt request from PLB-to-OPB bridge 4 Interrupt request from PLB-to-AHB bridge 5 Interrupt request from DDR SDRAM HB 6 Interrupt request from PE1 or SRIO 7 Interrupt request from PE0	Master 4
8:9		Reserved	
10:15	PE0IRQ	PE0 Interrupt Request Bit: 10 Interrupt request from DDR SDRAM LL 11 Interrupt request from PLB-to-OPB bridge 12 Interrupt request from PLB-to-AHB bridge 13 Interrupt request from DDR SDRAM HB 14 Interrupt request from PE1 or SRIO 15 Interrupt request from PE0	Master 5
16:17		Reserved	
18:23	DMA0IRQ	DMA2P40 Interrupt Request Bit: 18 Interrupt request from DDR SDRAM LL 19 Interrupt request from PLB-to-OPB bridge 20 Interrupt request from PLB-to-AHB bridge 21 Interrupt request from DDR SDRAM HB 22 Interrupt request from PE1 or SRIO 23 Interrupt request from PE0	Master 6
24:25		Reserved	
26:31	HSDMAIRQ	HSDMA Interrupt Request Bit: 26 Interrupt request from DDR SDRAM LL 27 Interrupt request from PLB-to-OPB bridge 28 Interrupt request from PLB-to-AHB bridge 29 Interrupt request from DDR SDRAM HB 30 Interrupt request from PE1 or SRIO 31 Interrupt request from PE0	Master 7

2.1.3.6 PLB4 Master Interrupt Request Register 2 (SDR0_MIRQ2)

This is a 32-bit read/write register that controls interrupt requests from PLB masters 8 to 11.

Reset value = 0 for all fields.

Figure 2-6. Master Interrupt Request Register 2 (SDR0_MIRQ2)

0:1		Reserved	
2:7	I2OIRQ	I2O/FIFO Interrupt Request Bit: 2 Interrupt request from DDR SDRAM LL 3 Interrupt request from PLB-to-OPB bridge 4 Interrupt request from PLB-to-AHB bridge 5 Interrupt request from DDR SDRAM HB 6 Interrupt request from PE1 or SRIO 7 Interrupt request from PE0	Master 8
8:9		Reserved	
10:15	EIP94IRQ	EIP94 Interrupt Request Bit: 10 Interrupt request from DDR SDRAM LL 11 Interrupt request from PLB-to-OPB bridge 12 Interrupt request from PLB-to-AHB bridge 13 Interrupt request from DDR SDRAM HB 14 Interrupt request from PE1 or SRIO 15 Interrupt request from PE0	Master 9
16:17		Reserved	
18:23	MALIRQ	MAL Interrupt Request Bit: 10 Interrupt request from DDR SDRAM LL 11 Interrupt request from PLB-to-OPB bridge 12 Interrupt request from PLB-to-AHB bridge 13 Interrupt request from DDR SDRAM HB 14 Interrupt request from PE1 or SRIO 15 Interrupt request from PE0	Master 10
24:25		Reserved	
26:31	AHBIRQ	PLB4XAHB Interrupt Request Bit: 26 Interrupt request from DDR SDRAM LL 27 Interrupt request from PLB-to-OPB bridge 28 Interrupt request from PLB-to-AHB bridge 29 Interrupt request from DDR SDRAM HB 30 Interrupt request from PE1 or SRIO 31 Interrupt request from PE0	Master 11

2.1.3.7 PLB Slave Address Pipeline Register (SDR0_SLPIPE)

Reset value = 1 for all fields.

Figure 2-7. PLB Slave Address Pipeline Register (SDR0_SLPIPE)

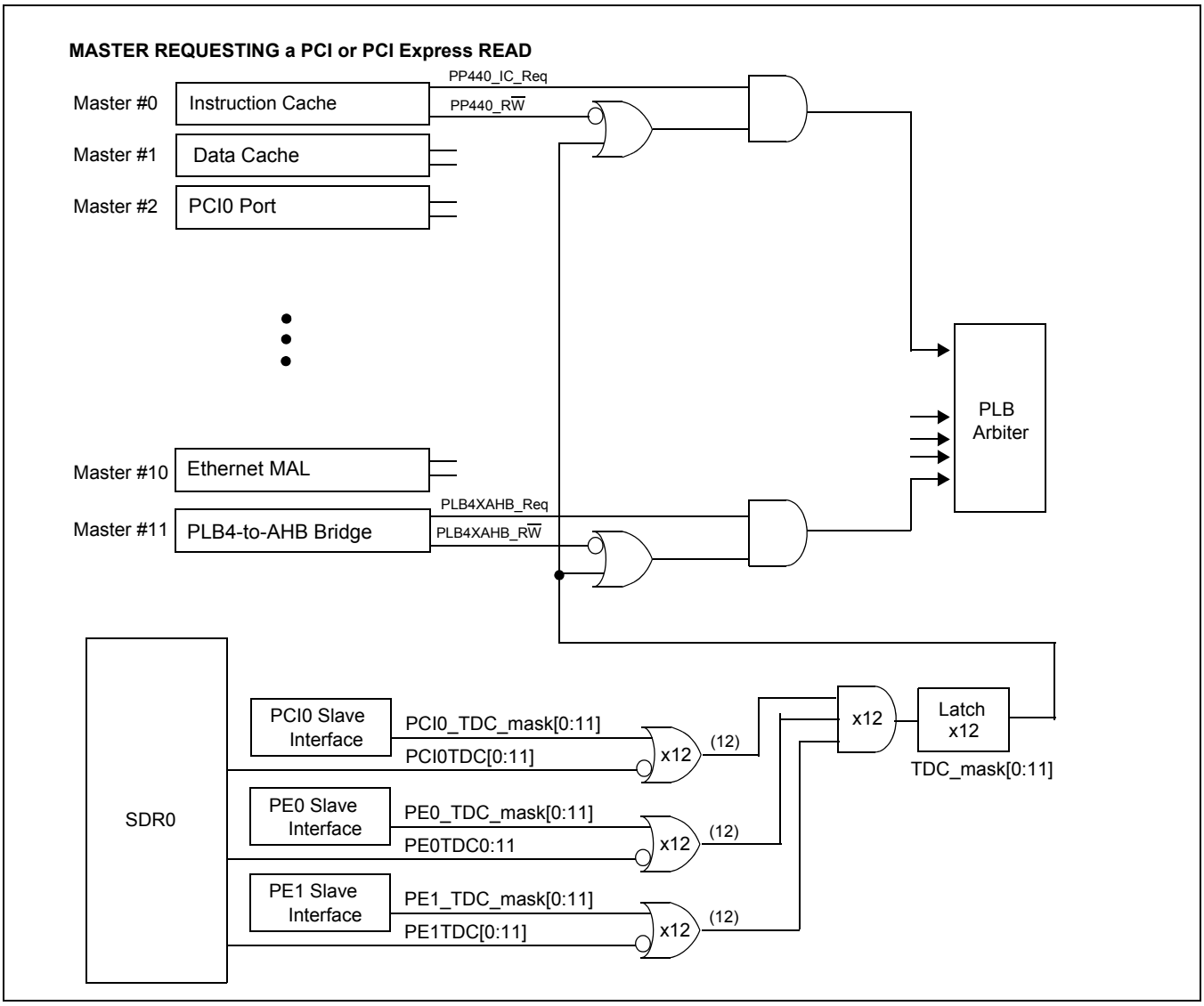
0	SRAP	SRAM Address Pipeline 0 SRAM address pipeline disabled 1 SRAM address pipeline enabled	
1	DSHBAP	DDR SDRAM HB Address Pipeline 0 DDR SDRAM HB address pipeline disabled 1 DDR SDRAM HB address pipeline enabled	
2	POBAP	PLB to OPB Bridge Address Pipeline 0 PLB to OPB bridge address pipeline disabled 1 PLB to OPB bridge address pipeline enabled	
3	DMAAP	I2O DMA Address Pipeline 0 I2O DMA address pipeline disabled 1 I2O DMA address pipeline enabled	
4	DSLLAP	DDR SDRAM LL Address Pipeline 0 DDR SDRAM LL address pipeline disabled 1 DDR SDRAM LL address pipeline enabled	
5	PCI0AP	PCI0 Address Pipeline 0 PCI0 address pipeline disabled 1 PCI0 address pipeline enabled	
6	OCMAP	OCM Address Pipeline 0 OCM address pipeline disabled 1 OCM address pipeline enabled	
7:31		Reserved	

2.1.4 Target Directed Completion

PCI or PCI Express ports implement a delayed read policy for read operations on the PLB bus. Thus, if a read operation targeted to a PCI or PCI Express port is performed, instead of holding the PLB bus with wait states until the read data arrives from the PCI or PCI Express link (this could be a long time) the port queues the transaction within itself and re-arbitrates the PLB bus, ending the transaction on the PLB.

The PLB arbiter is now free to grant the PLB bus to any other master that is requesting it, thus allowing other PLB traffic while the read data is arriving from the PCI or PCI Express link. Eventually, the PLB arbiter will grant back to the originating master. When the read data is available within the PCI or PCI Express port, then the PLB is requested again. If the link latency is large because of heavy usage on PCI or PCI Express, numerous rearbitration cycles can occur before the master finally gets the read operation acknowledgement and data is returned.

Figure 2-8. Target Directed Completion Capability



The Target Directed Completion (TDC) feature enables the PLB slave interface of a PCI or PCI Express port, during an outbound read operation, to notify the originating PLB master when the data is available in the PCI or PCI port. Thus, the originating master does not have to keep polling the PCI or PCI Express port. As a result, all or most of the PLB rearbitration cycles during which the master is asking for data that has not arrived yet, are eliminated.

The master performs a read operation, which initially is rearbitrated and queued with the PCI or PCI Express port. Then the PCI or PCI Express port asserts a TDC_mask (active low signal) for that specific master (one bit per master available in the PPC460EX/EXr/GT) until the read data has arrived. The TDC_mask gates off the respective master's request to the PLB arbiter, effectively preventing the master from running the transaction until the data becomes available. When the data becomes available, the mask is deasserted and the master's transaction is completed on the PLB bus.

2.1.5 Target Directed Completion Register for PCI Express

Table 2-4 shows the register that controls TDC.

Table 2-4. PCI Express Target Directed Completion Register

Mnemonic	Register	Address	Access	Page
SDR0_TDC1	PCI Target Directed Completion Setting 1 Register	0x0270	R/W	113
SDR0_TDC2	PCI Target Directed Completion Setting 2 Register	0x0271	R/W	114

2.1.5.1 PCI Target Directed Completion Setting 1 (SDR0_TDC1)

Reset value = 1 for all non-reserved fields.

Figure 2-9. PCI Target Directed Completion Setting 1 (SDR0_TDC1)

0	PCI0TDC0	Target Directed Completion Enable mode during read operation of the PCI0 port by the PPC440 Instruction-Cache: 0 Disabled. 1 Enabled.	Master 0 The 0/1 state is the same for all bits in this register.
1	PCI0TDC1	Target Directed Completion Enable mode during read operation of the PCI0 port by the PPC440 Data-Cache.	Master 1
2	PCI0TDC2	Target Directed Completion Enable mode during read operation of the PCI0 port by the PCI0.	Master 2
3	PCI0TDC3	Target Directed Completion Enable mode during read operation of the PCI0 port by the PCI Express PE1 or SRIO.	Master 3
4		Reserved	
5	PCI0TDC5	Target Directed Completion Enable mode during read operation of the PCI0 port by the PCI Express PE0.	Master 5
6	PCI0TDC6	Target Directed Completion Enable mode during read operation of the PCI0 port by the DMA2P40.	Master 6
7	PCI0TDC7	Target Directed Completion Enable mode during read operation of the PCI0 port by the HSDMA.	Master 7
8	PCI0TDC8	Target Directed Completion Enable mode during read operation of the PCI0 port by the I2O/FIFO.	Master 8
9	PCI0TDC9	Target Directed Completion Enable mode during read operation of the PCI0 port by the EIP94.	Master 9
10	PCI0TDC10	Target Directed Completion Enable mode during read operation of the PCI0 port by the MAL.	Master 10
11	PCI0TDC11	Target Directed Completion Enable mode during read operation of the PCI0 port by the PLB4XAHB bridge.	Master 11
12:15		Reserved	
16	PE0TDC0	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the PPC440 Instruction-Cache.	Master 0

17	PE0TDC1	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the PPC440 Data-Cache.	Master 1
18	PE0TDC2	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the PCI0.	Master 2
19	PE0TDC3	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the PCI Express PE1 or SRIO.	Master 3
20		Reserved	
21	PE0TDC5	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the PCI Express PE0.	Master 5
22	PE0TDC6	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the DMA2P40.	Master 6
23	PE0TDC7	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the HSDMA.	Master 7
24	PE0TDC8	Target Directed Completion Enable mode during read operation of the PCI0 port by the I2O/FIFO.	Master 8
25	PE0TDC9	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the EIP94.	Master 9
26	PE0TDC10	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the MAL.	Master 10
27	PE0TDC11	Target Directed Completion Enable mode during read operation of the PCI Express PE0 port by the PLB4XAHB.	Master 11
28:31		Reserved	

2.1.5.2 PCI Target Directed Completion Setting 2 (SDR0_TDC2)

Reset value = 1 for all non-reserved fields.

Figure 2-10. PCI Target Directed Completion Setting 2 (SDR0_TDC2)

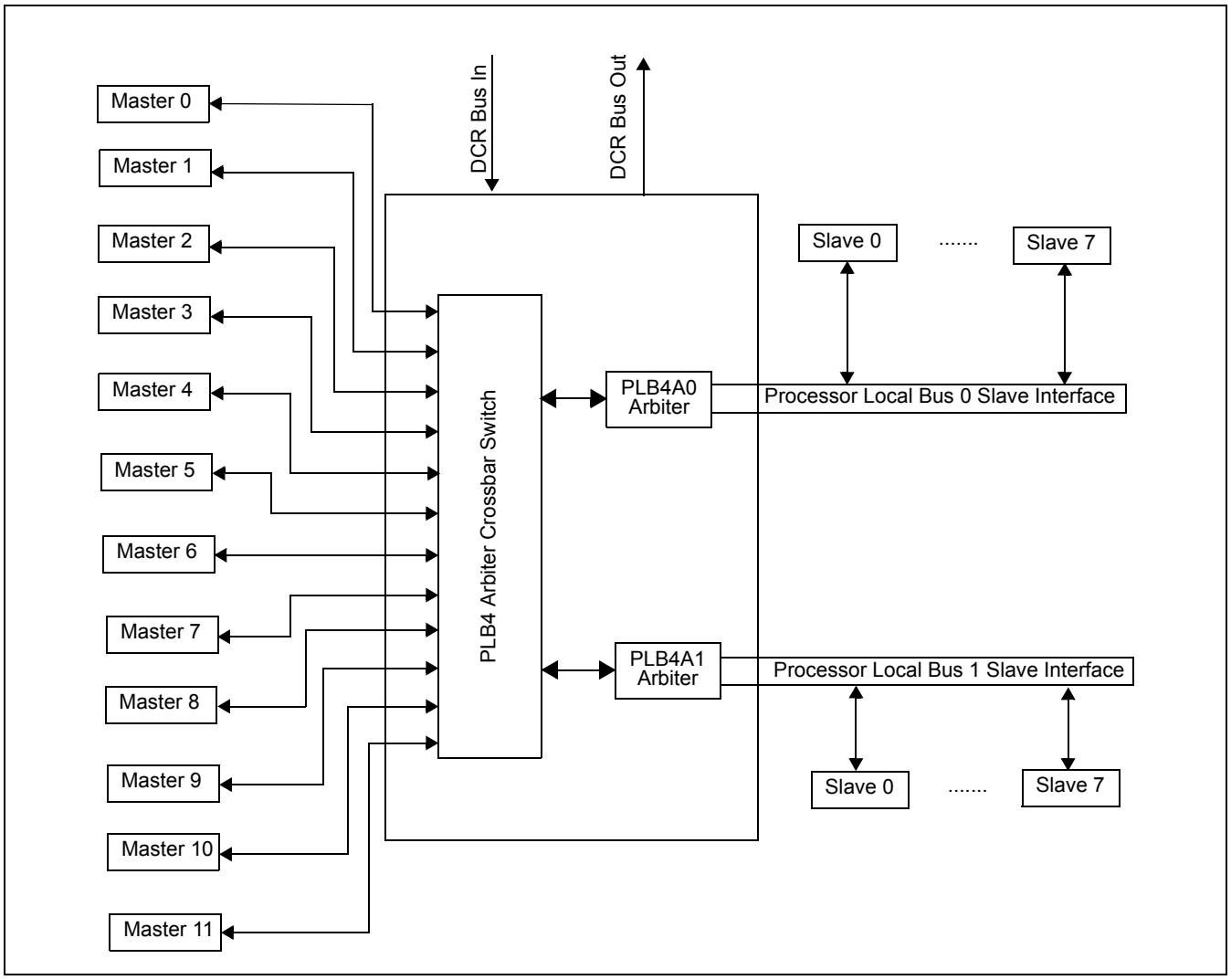
0	PE1TDC0	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the PPC440 Instruction-Cache: 0 TDC disabled. 1 TDC enabled.	Master 0 The 0/1 state is the same for all bits in this register.
1	PE1TDC1	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the PPC440 Data-Cache.	Master 1
2	PE1TDC2	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the PCI0.	Master 2
3	PE1TDC3	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the PCI Express PE1.	Master 3
4		Reserved	

5	PE1TDC5	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the PCI Express PE0.	Master 5
6	PE1TDC6	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the DMA2P40.	Master 6
7	PE1TDC7	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the HSDMA.	Master 7
8	PE1TDC8	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the I2O/FIFO.	Master 8
9	PE1TDC9	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the EIP94.	Master 9
10	PE1TDC10	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the MAL.	Master 10
11	PE1TDC11	Target Directed Completion Enable mode during read operation of the PCI Express PE1 or SRIO port by the PLB4XAHB.	Master 11
12:31		Reserved	

2.1.6 PLB4 Arbiter

This is the high performance128-bit version crossbar switch supporting two processor local bus arbiters implementing the three cycle acknowledge timing protocol. It is a soft core consisting of a 12-master to 2-PLB slave segments crossbar switch including for each PLB segment: a bus arbitration control unit, a watchdog timer, address path, write data path, and read data path units.

Figure 2-11. PLB Crossbar Arbiter Interconnection



As shown in *Figure 2-11*, the on-chip bus structure provides a link between PLB bus masters such as the processor, DMA controller, OPB to PLB bridge, and other PLB master devices and PLB bus slaves such as the memory controller, PLB to OPB bridge, and other PLB slave devices, residing on either the PLB0 slave interface (Low Latency segment) or the PLB1 slave interface (High Bandwidth segment).

The processor local bus (PLB) is the high performance bus used to access memory through the bus interface units. This PLB is implemented with a crossbar switch allowing separate PLB master devices to independently access slave devices attached to PLB 0 slave interface and PLB1 slave Interface concurrently for improved PLB bus performance.

The device control register (DCR) bus is used primarily for accessing status and control registers. It is meant to off-load the PLB from the lower performance status and control read and write transfers. The DCR bus architecture allows data transfers among peripherals to occur independently from, and concurrent with, data transfers between the PLB masters and PLB slaves.

2.1.6.1 PLB4 Arbiters 0 and 1 Registers

PLB4 arbiter registers listed in *Table 2-5* are DCRs accessed using the **mfdcr** and **mtdcr** instructions.

Table 2-5. PLB4 Arbiters 0 and 1 Registers

Mnemonic	Register Name	DCR Address	Access	Page
PLB4A_REVID	PLB4 Arbiter Revision ID Register	0x0080	R	117
PLB4A0_ACR	PLB4A0 Arbiter Control Register	0x0081	R/W	118
PLB4A0_ESRL	PLB4A0 Error Status Register Low	0x0082	R/Clear	121
PLB4A0_ESRH	PLB4A0 Error Status Register High	0x0083	R/Clear	123
PLB4A0_EARL	PLB4A0 Error Address Register Low	0x0084	R	124
PLB4A0_EARH	PLB4A0 Error Address Register High	0x0085	R	124
PLB4A0_ESRL*	PLB4A0 Error Status Register Low (*reserved for diagnostic use only)	0x0086	Set(W)	121
PLB4A0_ESRH*	PLB4A0 Error Status Register High (*reserved for diagnostic use only)	0x0087	Set(W)	121
PLB4A_CCR	PLB4 Crossbar Control Register	0x0088	R/W	125
PLB4A1_ACR	PLB4A1 Arbiter Control Register	0x0089	R/W	118
PLB4A1_ESRL	PLB4A1 Error Status Register Low	0x008A	R/Clear	121
PLB4A1_ESRH	PLB4A1 Error Status Register High	0x008B	R/Clear	123
PLB4A1_EARL	PLB4A1 Error Address Register Low	0x008C	R	124
PLB4A1_EARH	PLB4A1 Error Address Register High	0x008D	R	124
PLB4A1_ESRL*	PLB4A1 Error Status Register Low (*reserved for diagnostic use only)	0x008E	Set(W)	121
PLB4A1_ESRH*	PLB4A1 Error Status Register High (*reserved for diagnostic use only)	0x008F	Set(W)	123

2.1.6.2 PLB4 Arbiter Revision ID Register (PLB4A_REVID)

PLB4A_REVID is a 32-bit read-only register that contains the revision ID of the PLB4 arbiter. The contents of the register can be accessed by using the move from device control register (**mfdcr**) instruction. The PLB4A_REVID register can be accessed with CPU_dcrAddr6:9 = 0x2. The register is not affected by reset.

Figure 2-12. PLB4A Arbiter Revision ID Register (PLB4A_REVID)

0:11		Reserved	
12:23	RN	Revision number	Corresponds to the RCS revision of the source RTL
24:31	BRN	Branch revision number	Corresponds to the RCS branch revision of the source RTL

2.1.6.3 PLB4 Arbiter Control Register (PLB4An_ACR)

PLB4An_ACR is a 32-bit register which controls the modes of operation for the arbiter. The priority mode, high bus utilization, read pipeline enable, and write pipeline enable are contained in this register. At reset, Fair priority mode for all priority levels is selected, high bus utilization is enabled, and two deep read and write pipelining are enabled. PLB4An_ACR[0:31] = 0xDB000000.

Figure 2-13. PLB4 Arbiter Control Register (PLB4An_ACR)

0:3	PPM	PLB Priority Mode PPM0 0 Priority level 00 Fixed priority. 1 Priority level 00 Fair priority. PPM1 0 Priority level 01 Fixed priority 1 Priority level 01 Fair priority PPM2 - Reserved (always zero) PPM3 0 Priority level 11 Fixed priority 1 Priority level 11 Fair priority	
4	HBU	High Bus Utilization 0 Disabled 1 Enabled	If read and write pipelining are disabled this feature has no effect on arbiter operation.
5:6	RDP	Read Pipeline Control 00 Read pipelining disabled 01 2 Deep read pipe 10 3 Deep read pipe 11 4 Deep read pipe	
7	WRP	Write Pipeline Control 0 Write pipeline disabled 1 2 Deep write pipe	
8:31		Reserved	

The following sections provide details on the effect of the bit settings in PLB4An_ACR.

PPM Field

During the bus arbitration cycle, the bus arbitration control unit uses the Mn_priority0:1 signals to determine which master will be granted the bus. The priority inputs of masters with their respective Mn_request signal asserted are used in determine the highest request priority. In addition, the core supports the fixed priority and the fair priority scheme to handle "tie" situations (that is, situations when two or more masters request the bus simultaneously while presenting the same level of request priority). The selection of the priority mode during tie situations is controlled by these bits.

Under the fixed priority scheme, each bus master is assigned a unique priority level as shown below. Note that only three distinct levels of priority are supported by the Crossbar Arbiter. A master requesting on Priority Level 2 is treated by the Crossbar Arbiter as if it had requested on Priority Level 1. *Table 2-6* shows all 12 bus masters in the order of priority:

Table 2-6. Master Priority Orders

Highest Priority	Decreasing Priority ----->										Lowest Priority
For Priority Level 3 (highest):											
M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
For Priority Level 2:											
Merged with priority level 1											
For Priority Level 1:											
M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
For Priority Level 0 (lowest):											
M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11

Similarly, under the fair priority scheme, each bus master is assigned a unique priority level as shown above. However, once one master at a particular Mn_priority0:1 level is granted the bus, all other masters requesting the bus at that priority level at that time that continue to present their request at that same priority level will eventually be granted the bus before any other master (including the one originally granted the bus) is subsequently granted the bus at that priority level.

Once the last master that had not been granted the bus which is still requesting the bus on that priority level is granted the bus (or if no masters that had not been granted the bus continue to request the bus at that priority level), arbitration for the next master will be open to all requesting masters at that priority level again using the original priority above. If two masters request the bus simultaneously and continuously at the same priority level each master will be granted the bus 50% of the time at that priority level. If three masters request the bus simultaneously and continuously at the same priority level, each master will be granted the bus 33% of the time at that priority level. If four masters request the bus simultaneously and continuously at the same priority level, each master will be granted the bus 25% of the time at that priority level, etc.

This assumes all masters assert and continue to assert the same request priority. Note that a "qualified" request is a Mn_Request addressed to this PLB segment which is not otherwise blocked (for example, by the same master having a "like transaction" in progress on the other PLB segment).

The detailed manner in which this fairness algorithm works is as follows:

- For the PLB4An arbiter, there is one "fairness request" latch for each of the 12 masters on each of the three distinct priority levels (for a total of 36 latches for the PLB4An arbiter).
- If fairness on a particular level is *not* enabled, then the latches for that level are ignored (treated as if they were set to 0).
- If fairness on a particular level is enabled and *all* latches on that level are set to 0 (the initial condition), then on the arbitration cycle, arbitration on that level occurs among *all* masters with valid "qualified" requests active on that arbitration level. In the same clock cycle, the "fairness request" latches corresponding to each master with a valid "qualified" request on that arbitration level are set to one by the next clock rising edge.
- If fairness on a particular level is enabled and at least one latch on that level is set to one, then arbitration on that level occurs *only* among those masters with a valid "qualified" request on that arbitration level whose "fairness request" latch is set to one by the next clock rising edge.
- On *any* clock cycle, if a master stops presenting a valid "qualified" request on a particular arbitration level, then that master's "fairness request" latch is cleared to zero by the next clock rising edge.

- On *any* clock cycle, if arbAddrSelRegn becomes active (indicating that Master n has "won" the current arbitration), then each "fairness request" latch associated with Master n is cleared to 0 by the next rising clock edge.

In this manner, all sustained "qualified" requests which have been latched in the "fairness request" latches on the same arbitration level must be satisfied once for PLB4An before any master's request on the same level may be granted a second time on PLB4An. Note that a "qualified" request on a particular arbitration level may be removed without being granted as a result of the master dynamically changing the requested arbitration level or by the master aborting the request.

In this example, when arbAddrSelRegn is a 1 this indicates that master n has been latched as winner of the current arbitration on PLB4An (until terminated after a slave responds with addrack or rearbitrate, or the arbiter's watchdog timer issues a timeout).

HBU Bit

This bit enables a feature that will ensure both the read and write data buses are always busy if there are pending master requests for both buses. An additional requirement is that there are no bus lock requests pending or bus locked transactions in progress. These conditions will temporarily disable this feature. When this bit is 1 the arbiter will promote a lower priority request to the current active request on the address bus under the following conditions, (there are two cases, one for each data bus):

- If the read data bus is busy with a transfer and a secondary read request has been address acknowledged (addrAcked) and there are additional read request(s) pending, and the next highest request in the arbitration queue is a read request. Then a lower priority master write request will be promoted to the current active write bus request on the address bus if the write data bus is idle.
- A similar case also exists for the write bus being busy with both a primary request active and a secondary request address acknowledged (addrAcked) and the next highest request in the arbitration queue is a write request also. If a lower priority read request is pending and the read data bus is idle then the lower priority read request will be promoted to the active request on the address bus.

If read and write pipelining are disabled this feature will have no affect on arbiter operation. The default mode after reset is enabled. Setting this bit to 0 will disable this feature. It is recommended this bit be cleared immediately after reset.

RDP Field

These bits control the depth of read pipelining. That is the number of outstanding read requests broadcast to the slaves. When PLB4An_ACR[RDP] = 01 the arbiter will default to a two deep read pipeline after reset. This allows the arbiter to broadcast a primary and secondary read to the slaves. When PLB4An_ACR[RDP] = 10 the arbiter is capable of generating a primary, secondary, and third read to the slaves. When PLB4An_ACR[RDP] = 11 the arbiter is capable of generating a primary, secondary, third, and fourth read to the slaves. Clearing these bits disables read pipelining and only primary read transfers are broadcast. In this case PLB0_SAValid will never be asserted for a read request. The arbiter has eight separate slave address acknowledge signals, Sl_PLB0_addrAck0:7. This allows the arbiter to determine which slave is responding to the PLB0_SAValid signal assertion. The arbiter also has eight separate read primary signals, PLB0_rdPrim0:7, which should be connected to the corresponding slave. Consequently upon completion of the primary transfer the arbiter will notify the appropriate slave that its pipelined transfer is now the primary and may begin driving the data bus and controls two cycles following the assertion of PLB0_rdPrimn.

WRP Field

This bit controls the depth of write pipelining. That is the number of outstanding write requests broadcast to the slaves. When PLB0_ACR[WRP] = 1 the arbiter defaults to a two deep write pipeline after reset. This allows the arbiter to broadcast a primary and secondary write to the slaves. When PLB0_ACR[WRP] = 0 write pipelining will be disabled and only primary write transfers will be broadcast. In this case PLB0_SAValid will never be asserted for a write request.

2.1.6.4 PLB4 Error Status Register Low (PLB4An_ESRL)

This register identifies time-out errors on PLB4 bus transfers, the master initiating the transfer, and the type of transfer. Each PLB4An_ESRL[PTEn] field (n in the field is the master ID) can be locked by the master. Once locked, PLB4An_ESRL[PTEn] fields cannot be updated if a subsequent error occurs until the corresponding PLB4An_ESRL[FLKn] field is cleared. To clear a PLB4An_ESRL field, write 1 to the field. Writing 0 to a PLB4A0_ESRL field does not affect the field.

The PLB4A0_ESRL register can be accessed with DCR address = 0x82 (read/clear) and DCR address = 0x86 (set). The PLB4A1_ESRL can be accessed with DCR address = 0x8A (read/clear) and DCR address = 0x8E (set). At reset, all bits in the PLB4An_ESRL are loaded with zeroes. The registers at DCR = 0x86 and 0x8E are used only for diagnostic purposes.

Figure 2-14. PLB4 Error Status Register Low (PLB4An_ESRL)

0	PTE0	Master 0 PLB Timeout Error Status 0 No master timeout error 1 Master timeout error	Master 0 - Processor instruction cache unit (ICU)
1	R/W0	Master 0 Read/Write Status 0 Master error operation was a write 1 Master ICU error operation was a read	
2	FLK0	Master 0 PLB4An_ESR Field Lock 0 Master field is unlocked 1 Master field is locked	
3	ALK0	Master 0 PLB4An_EAR Address Lock 0 Master address is unlocked 1 Master address is locked	
4	PTE1	Master 1 PLB Timeout Error Status 0 No master timeout error 1 Master timeout error	Master 1 - Processor data cache read unit (DCU)
5	R/W1	Master 1 Read/Write Status 0 Master error operation was a write 1 Master error operation was a read	
6	FLK1	Master 1 PLB4An_ESR Field Lock 0 Master field is unlocked 1 Master field is locked	
7	ALK1	Master 1 PLB4An_EAR Address Lock 0 Master address is unlocked 1 Master address is locked	
8	PTE2	Master 2 PLB Timeout Error Status 0 No master timeout error 1 Master timeout error	Master 2 - PCI0
9	R/W2	Master 2 Read/Write Status 0 Master error operation was a write 1 Master error operation was a read	

10	FLK2	Master 2 PLB4An_ESR Field Lock 0 Master field is unlocked 1 Master field is locked	
11	ALK2	Master 2 PLB4An_EAR Address Lock 0 Master address is unlocked 1 Master address is locked	
12	PTE3	Master 3 PLB Timeout Error Status 0 No Master timeout error 1 Master timeout error	Master 3 - PCIE/SRIO
13	R/W3	Master 3 Read/Write Status 0 Master error operation was a write 1 Master error operation was a read	
14	FLK3	Master 3 PLB4An_ESR Field Lock 0 Master field is unlocked 1 Master field is locked	
15	ALK3	Master 3 PLB4An_EAR Address Lock 0 Master address is unlocked 1 Master address is locked	
16	PTE4	Master 4 PLB Timeout Error Status 0 No Master timeout error 1 Master timeout error	Master 4 - Processor data cache write unit (DCU)
17	R/W4	Master 4 Read/Write Status 0 Master error operation was a write 1 Master error operation was a read	
18	FLK4	Master 4 PLB4An_ESR Field Lock 0 Master field is unlocked 1 Master field is locked	
19	ALK4	Master 4 PLB4An_EAR Address Lock 0 Master address is unlocked 1 Master address is locked	
20	PTE5	Master 5 PLB Timeout Error Status 0 No Master timeout error 1 Master timeout error	Master 5 - PCIE0
21	R/W5	Master 5 Read/Write Status 0 Master error operation was a write 1 Master error operation was a read	
22	FLK5	Master 5 PLB4An_ESR Field Lock 0 Master field is unlocked 1 Master field is locked	
23	ALK5	Master 5 PLB4An_EAR Address Lock 0 Master address is unlocked 1 Master address is locked	
24	PTE6	Master 6 PLB Timeout Error Status 0 No Master timeout error 1 Master timeout error	Master 6 - DMA2P40
25	R/W6	Master 6 Read/Write Status 0 Master error operation was a write 1 Master error operation was a read	
26	FLK6	Master 6 PLB4An_ESR Field Lock 0 Master PLB4An_ESR field is unlocked 1 Master PLB4An_ESR field is locked	
27	ALK6	Master 6 PLB4An_EAR Address Lock 0 Master 6 address is unlocked 1 Master 6 PLB4An_EAR is locked	

28	PTE7	Master 7 PLB Timeout Error Status 0 No Master timeout error 1 Master timeout error	Master 7 - HSDMA
29	R/W7	Master 7 Read/Write Status 0 Master error operation was a write 1 Master error operation was a read	
30	FLK7	Master 7 PLB4An_ESR Field Lock 0 Master field is unlocked 1 Master field is locked	
31	ALK7	Master 7 PLB4An_EAR Address Lock 0 Master address is unlocked 1 Master address is locked	

2.1.6.5 PLB4 Error Status Register High (PLB4An_ESRH)

This register described in the following figure identifies time-out errors on PLB4 bus transfers, the master initiating the transfer, and the type of transfer. Each PLB4An_ESRH[PTEn] field (n in the field is the master ID) can be locked by the master. Once locked, PLB4An_ESRH[PTEn] fields cannot be updated if a subsequent error occurs until the corresponding PLB4An_ESRH[FLKn] field is cleared. To clear a PLB4An_ESRH field, write 1 to the field. Writing 0 to a PLB4A0_ESRH field does not affect the field.

The PLB4A0_ESRH register can be accessed with DCR address = 0x83 (read/clear) and DCR address = 0x87 (set). The PLB4A1_ESRH can be accessed with DCR address = 0x8B (read/clear) and DCR address = 0x8F (set). At reset, all bits in the PLB4An_ESRH are loaded with zeroes. The registers at DCR = 0x86 and 0x8E are for diagnostic purposes only.

Figure 2-15. PLB4 Error Status Register High (PLB4An_ESRH)

0	PTE8	Master 8 PLB Timeout Error Status 0 No timeout error 1 Timeout error	Master 8 -I2ODMA
1	R/W8	Master 8 Read/Write Status 0 Error operation was a write 1 ICU error operation was a read	
2	FLK8	Master 8 PLB4An_ESR Field Lock 0 Field is unlocked 1 Field is locked	
3	ALK8	Master 8 PLB4An_EAR Address Lock 0 Address is unlocked 1 Address is locked	
4	PTE9	Master 9 PLB Timeout Error Status 0 No timeout error 1 Timeout error	Master 9 - Security (EIP94)
5	R/W9	Master 9 Read/Write Status 0 Error operation was a write 1 Error operation was a read	
6	FLK9	Master 9 PLB4An_ESR Field Lock 0 Unlocked 1 Locked	
7	ALK9	Master 9 PLB4An_EAR Address Lock 0 Unlocked 1 Locked	

8	PTE10	Master 10 PLB Timeout Error Status 0 No timeout error 1 1 Timeout error	Master 10 - MAL
9	R/W10	Master 10 Read/Write Status 0 Error operation was a write 1 Error operation was a read	
10	FLK10	Master 10 PLB4An_ESR Field Lock 0 Field is unlocked 1 Field is locked	
11	ALK10	Master 10 PLB4An_EAR Address Lock 0 Address is unlocked 1 Address is locked	
12	PTE11	Master 11 PLB Timeout Error Status 0 No master timeout error 1 Master timeout error	Master 11 - MAL
13	R/W11	Master 11 Read/Write Status 0 Error operation was a write 1 Error operation was a read	
14	FLK11	Master 11 PLB4An_ESR Field Lock 0 Unlocked 1 Locked	
15	ALK11	Master 11 PLB4An_EAR Address Lock 0 Unlocked 1 Locked	
16:31		Reserved	

2.1.6.6 PLB4 Error Address Register Low (PLB4An_EARL)

The read-only PLB4An_EARL register contains the lower 32 bits of the address of the access on which a bus time-out error occurred. The PLB4An_EARL can be locked by the master. Once locked, the PLB4An_EARL cannot be updated, if a subsequent error occurs, until all PLB4An_ESRL[FLCKn] and PLB4An_ESRH[FLCKn] fields are cleared (n is the master ID).

The PLB4An_EARL is not affected by Reset.

Figure 2-16. PLB4 Error Address Register Low (PLB4An_EARL)

0:31		PLB4An lower address of bus timeout error	

2.1.6.7 PLB4 Error Address Register High (PLB4An_EARH)

The read-only PLB4An_EARH register contains the upper 32 bits of the address of the access on which a bus time-out error occurred. The PLB4An_EARH can be locked by the master. Once locked, the PLB4An_EARH cannot be updated, if a subsequent error occurs, until all PLB4An_ESRL[FLCKn] and PLB4An_ESRH[FLCKn] fields are cleared (n is the master ID).

The PLB4An_EARH is not affected by Reset.

Figure 2-17. PLB4 Error Address Register High (PLB4An_EARH)

0:31		PLB4An upper address of bus timeout error

2.1.6.8 PLB4 Crossbar Control Register (PLB4A_CCR)

The PLB4 Crossbar Control Register (PLB4A_CCR) controls the modes of operation for the crossbar switch. The PLB Segmentation Address Decode selection is contained in this register. At reset PLB4A_CCR[PSA] is loaded with the value of PGM_PLBSegAddr0:3.

Figure 2-18. PLB4 Crossbar Control Register (PLB4A_CCR)

0:3	PSA	PLB Segmentation Upper Address Control See *Table 2-7*	These bits, in combination with Mn_SegABus0:3, determine which slave PLB segment a Master's Mn_request is routed to by the crossbar switch. These bits must be set to 0x8.
4:31		Reserved	

Each bit of the PSA field determines whether the corresponding bit of Mn_SegABus0:3 will be used in determining whether to access the PLB4A0 or the PLB4A1 slave segment. If *all* of the Mn_SegABus0:3 bits for which the corresponding bit of PSA is a 1 are also 1s, then PLB4A1 is accessed. Otherwise PLB4A0 is accessed. *Table 2-7* shows how PSA bits are used with Mn_SegAbus0:3 to map master accesses to either PLB4A0 or PLB4A1.

Table 2-7. PLB Segment Access

PSA0:3	Mn_SegAbus0:3		Access PLB Segment
	From	To	
1000	0000	0111	PLB4A0
1000	1000	1111	PLB4A1

2.1.7 PLB4 to OPB Bridge Registers

The PLB4 to OPB bridge registers listed in the following figure are DCRs accessed using the **mfdcr** and **mtdcr** instructions.

Table 2-8. PLB4 to OPB Bridge Registers

Mnemonic	Register Name	DCR Address	Access	Page
PLB42OPB0_BESR0	PLB4 to OPB0 Bridge Error Status Register 0 (Master IDs 0, 1, 2, and 3)	0x090	R/Clear	126
PLB42OPB0_BEARL	PLB4 to OPB0 Bridge Error Address Register Low	0x092	R/O	127
PLB42OPB0_BEARH	PLB4 to OPB0 Bridge Error Upper Address Register High	0x093	R/O	128
PLB42OPB0_BESR1	PLB4 to OPB0 Bridge Error Status Register 1 (Master IDs 4, 5, 6, and 7)	0x094	R/Clear	128
PLB42OPB0_CFG	PLB4 to OPB0 Bridge Configuration Register	0x096	R/Clear	130
PLB42OPB0_LATENCY	PLB4 to OPB0 Bridge Burst Latency Timer	0x098	R/Clear	130
PLB42OPB0_REVID	PLB4 to OPB0 Bridge Revision ID Register	0x09A	R/O	130

2.1.7.1 PLB4 to OPB Bridge Error Status Register 0 (PLB42OPB0_BESR0)

The PLB4 to OPB bridge writes error information into the appropriate PLB42OPB0_BESRm register. (For master IDs 0, 1, 2, and 3, m = 0; for master IDs 4, 5, 6, and 7, m = 1.)

PLB42OPB0_BESRm fields can be locked using the PLB42OPB0_BESRm[FLKn] and PLB42OPB0_BESRm[ALKn] fields (n is the master ID). Once locked, the PLB42OPB0_BESRm fields associated with a master cannot be overwritten if a subsequent error occurs until the locking fields are cleared. To clear a lock, write 1 to the PLB42OPB0_BESRm[FLKn] and PLB42OPB0_BESRm[ALKn] fields that are set. Writing 0 to a lock field does not affect the field.

Figure 2-19. PLB4 to OPB Bridge Error Status Register 0 (PLB42OPB0_BESR0)

0:1	PTE0	PLB Timeout Error Status Master 0 00 No master 0 error occurred 01 Master 0 timeout error occurred 10 Master 0 slave error occurred 11 Reserved	PLB4 master 0 is the read-only instruction cache unit (ICU).
2	R/W0	Read Write Status Master 0 0 Master 0 error operation is a write 1 Master 0 error operation is a read	
3	FLK0	PLB42OPB0_BESR0 Field Lock Master 0 0 Master 0 PLB42OPB0_BESR0 field is unlocked 1 Master 0 PLB42OPB0_BESR0 field is locked	
4	ALK0	PLB42OPB0_BEAR Address Lock Master 0 0 Master 0 PLB42OPB0_BEAR address is unlocked 1 Master 0 PLB42OPB0_BEAR address is locked	
5	WIRQ0	Write Error Interrupt Master 0 0 No write error detected - master 0 interrupt request is inactive 1 Write error detected - master 0 interrupt request is active	
6:7		Reserved	
8:9	PTE1	PLB Timeout Error Status Master 1 00 No master 1 error occurred 01 Master 1 timeout error occurred 10 Master 1 slave error occurred 11 Reserved	PLB4 master 1 is the read-only data cache unit (DCU).
10	R/W1	Read/Write Status Master 1 0 Master 1 error operation is a write 1 Master 1 error operation is a read	
11	FLK1	PLB42OPB0_BESR0 Field Lock Master 1 0 Master 1 PLB42OPB0_BESR0 field is unlocked 1 Master 1 PLB42OPB0_BESR0 field is locked	
12	ALK1	PLB42OPB0_BEAR Address Lock Master 1 0 Master 1 PLB42OPB0_BEAR address is unlocked 1 Master 1 PLB42OPB0_BEAR address is locked	
13	WIRQ1	Write Error Interrupt Master 1 0 No write error detected - master 1 interrupt request is inactive 1 Write error detected - master 1 interrupt request is active	
14:15		Reserved	

16:17	PTE2	PLB Timeout Error Status Master 2 00 No master 2 error occurred 01 Master 2 timeout error occurred 10 Master 2 slave error occurred 11 Reserved	PLB4 master 2 is PCI0.
18	R/W2	Read/Write Status Master 2 0 Master 2 error operation is a write 1 Master 2 error operation is a read	
19	FLK2	PLB42OPB0PLB42OPB0_BESR0 Field Lock Master 2 0 Master 2 PLB42OPB0_BESR0 field is unlocked 1 Master 2 PLB42OPB0_BESR0 field is locked	
20	ALK2	PLB42OPB0_BEAR Address Lock Master 2 0 Master 2 PLB42OPB0_BEAR address is unlocked 1 Master 2 PLB42OPB0_BEAR address is locked	
21	WIRQ2	Write Error Interrupt Master 2 0 No write error detected - master 2 interrupt request is inactive 1 Write error detected - master 2 interrupt request is active	
22:23		Reserved	
24:25	PTE3	PLB Timeout Error Status Master 3 00 No master 3 error occurred 01 Master 3 timeout error occurred 10 Master 3 slave error occurred 11 Reserved	PLB4 master 3 PCIE1.
26	R/W3	Read/Write Status Master 3 0 Master 3 error operation is a write 1 Master 3 error operation is a read	
27	FLK3	PLB42OPB0_BESR0 Field Lock Master 3 0 Master 3 PLB42OPB0_BESR0 field is unlocked 1 Master 3 PLB42OPB0_BESR0 field is locked	
28	ALK3	PLB42OPB0_BEAR Address Lock Master 3 0 Master 3 PLB42OPB0_BEAR address is unlocked 1 Master 3 PLB42OPB0_BEAR address is locked	
29	WIRQ3	Write Error Interrupt Master 3 0 No write error detected - master 3 interrupt request is inactive 1 Write error detected - master 3 interrupt request is active	
30:31		Reserved	

2.1.7.2 PLB4 to OPB Bridge Error Address Register Low (PLB42OPB0_BEARL)

The read-only PLB42OPB0_BEARL reports the lower 32 bits of the address of a PLB4 to OPB transfer that results in an error. The PLB4 to OPB bridge writes the error address in the PLB42OPB0_BEARL, unless the associated PLB42OPB0_BESRm[ALCKn] field is set (m is either 0 or 1, depending on the master ID specified by n). Once locked, the PLB4 to OPB bridge cannot write PLB42OPB0_BEARL until all PLB42OPB0_BESRm[ALCKn] fields that are set are cleared.

Figure 2-20. PLB4 to OPB Bridge Error Address Register Low (PLB42OPB0_BEARL)

0:31	BEARL	Lower address of bus error	

2.1.7.3 PLB4 to OPB Bridge Error Address Register High (PLB42OPB0_BEARH)

The read-only PLB42OPB0_BEARL reports the lower 32 bits of the address of a PLB4 to OPB transfer that results in an error. The PLB4 to OPB bridge writes the error address in the PLB42OPB0_BEARL, unless the associated PLB42OPB0_BESRm[ALCKn] field is set (m is either 0 or 1, depending on the master ID specified by n). Once locked, the PLB4 to OPB bridge cannot write PLB42OPB0_BEARL until all PLB42OPB0_BESRm[ALCKn] fields that are set are cleared.

Figure 2-21. PLB4 to OPB Bridge Error Address Register High (PLB42OPB0_BEARH)

0:27		Reserved	
28:31	UA	Upper address of bus error	

2.1.7.4 PLB4 to OPB Bridge Error Status Register 1 (PLB42OPB0_BESR1)

Figure 2-22. PLB4 to OPB Bridge Error Status Register 1 (PLB42OPB0_BESR1)

0:1	PTE4	PLB Timeout Error Status Master 4 00 No error occurred 01 Timeout error occurred 10 Slave error occurred 11 Reserved	Master 4 write-only data cache unit (DCU).
2	R/W4	Read/Write Status Master 4 0 Error operation is a write 1 Error operation is a read	
3	FLK4	PLB42OPB0_BESR1 Field Lock Master 4 0 PLB42OPB0_BESR1 field is unlocked 1 PLB42OPB0_BESR1 field is locked	
4	ALK4	PLB42OPB0_BEAR Address Lock Master 4 0 PLB42OPB0_BEAR address is unlocked 1 PLB42OPB0_BEAR address is locked	
5	WIRQ4	Write Error Interrupt Master 4 0 No write error detected - interrupt request is inactive 1 Write error detected - interrupt request is active	
6:7		Reserved	
8:9	PTE5	PLB Timeout Error Status Master 5 00 No error occurred 01 Timeout error occurred 10 Slave error occurred 11 Reserved	Master 5 is PCIE0.
10	R/W5	Read/Write Status Master 5 0 Error operation is a write 1 Error operation is a read	
11	FLK5	PLB42OPB0_BESR1 Field Lock Master 5 0 PLB42OPB0_BESR1 field is unlocked 1 PLB42OPB0_BESR1 field is locked	
12	ALK5	PLB42OPB0_BEAR Address Lock Master 5 0 PLB42OPB0_BEAR address is unlocked 1 PLB42OPB0_BEAR address is locked	

13	WIRQ5	Write Error Interrupt Master 5 0 No write error detected - interrupt request is inactive 1 Write error detected - interrupt request is active	
14:15		Reserved	
16:17	PTE6	PLB Timeout Error Status Master 6 00 No error occurred 01 Timeout error occurred 10 Slave error occurred 11 Reserved	Master 6 is DMA2P40.
18	R/W6	Read/Write Status Master 6 0 Error operation is a write 1 Error operation is a read	
19	FLK6	PLB42OPB0_BESR1 Field Lock Master 6 0 PLB42OPB0_BESR1 field is unlocked 1 PLB42OPB0_BESR1 field is locked	
20	ALK6	PLB42OPB0_BEAR Address Lock Master 6 0 PLB42OPB0_BEAR address is unlocked 1 PLB42OPB0_BEAR address is locked	
21	WIRQ6	Write Error Interrupt Master 6 0 No write error detected - interrupt request is inactive 1 Write error detected - interrupt request is active	
22:23		Reserved	
24:25	PTE7	PLB Timeout Error Status Master 7 00 No error occurred 01 Timeout error occurred 10 Slave error occurred 11 Reserved	Master 7 is HSDMA.
26	R/W7	Read/Write Status Master 7 0 Error operation is a write 1 Error operation is a read	
27	FLK7	PLB42OPB0_BESR1 Field Lock Master 7 0 PLB42OPB0_BESR1 field is unlocked 1 PLB42OPB0_BESR1 field is locked	
28	ALK7	PLB42OPB0_BEAR Address Lock Master 7 0 PLB42OPB0_BEAR address is unlocked 1 PLB42OPB0_BEAR address is locked	
29	WIRQ7	Write Error Interrupt Master 7 0 No write error detected - interrupt request is inactive 1 Write error detected - interrupt request is active	
30:31		Reserved	

2.1.7.5 PLB4 to OPB Bridge Configuration Register (PLB42OPB0_CFG)

Figure 2-23. PLB4 to OPB Bridge Configuration Register (PLB42OPB0_CFG)		
0	ER	Enable rearbitration 0 PLB4 to OPB bridge rearbitration is enabled 1 PLB4 to OPB bridge rearbitration is disabled
1	L32R	Line 32 bit read 0 PLB4 to OPB bridge reads line operations at requested size of master 1 PLB4 to OPB bridge reads line operations at 32 bit slave size regardless of requesting master size
2:31		Reserved

2.1.7.6 PLB4 to OPB Bridge Burst Latency Timer (PLB42OPB0_LATENCY)

Figure 2-24. PLB4 to OPB Bridge Burst Latency Timer Register (PLB42OPB0_LATENCY)			
0	LE	Latency Enable 0 Latency timer disabled 1 Latency timer enabled	
1:4	LC	Latency Count 0000 - Minimum count value. 1111 - Maximum count value. PLB4 to OPB bridge will count 16 blocks of 16 OPB_xferAcks, (or 256 xferAcks) during a read or write burst sequence, and if OPB_pendReq is sampled high at the end of count - then the burst sequence is terminated.	When PLB42OPB0_LATENCY[LC = 0000] PLB4 to OPB bridge will count 16 OPB_xferAcks during a read or write burst sequence and if OPB_pendReq is sampled high at the end of the count - then the burst sequence is terminated. When PLB42OPB0_LATENCY[LC = 1111] PLB4 to OPB bridge will count 16 blocks of 16 OPB_xferAcks, (or 256 xferAcks) during a read or write burst sequence, and if OPB_pendReq is sampled high at the end of count - then the burst sequence is terminated.
5:31		Reserved	

2.1.7.7 PLB4 to OPB Bridge Revision ID (PLB42OPB0_REVID)

Figure 2-25. PLB4 to OPB Bridge Revision ID Register (PLB42OPB0_REVID)			
0:11		Reserved	
12:23	RN	Revision number	Corresponds to the RCS revision of the source RTL (hard wired to 0x1)
24:31	BRN	Branch revision number	Corresponds to the RCS branch revision of the source RTL (hard wired to 0x22)

2.2 On-Chip Peripheral Bus

The OPB attaches peripherals that do not have the high bandwidth or low latency requirements that would justify their direct attachment to the PLB. OPB segments interface to the PLB via the PLB to OPB bridge. This bridge allows PLB masters to access OPB slaves. Reverse bridges (OPB to PLB) allow OPB masters to access PLB slaves.

2.2.1 OPB Features

The on-chip peripheral bus features:

- A 32-bit address bus and a 32-bit data bus
- Dynamic bus sizing; byte, halfword, and fullword transfers
- Byte and halfword duplication for byte and halfword transfers
- Single-cycle transfer of data between OPB bus master and OPB slaves
- Sequential address (burst) protocol support
- A 16-cycle fixed bus time-out provided by the OPB arbiter
- Bus parking for reduced latency
- Bus arbitration overlapped with last cycle of bus transfers

2.2.2 OPB Master Assignments

Table 2-9 lists the two masters supported by OPBA0.

Table 2-9. Master Assignments

OPB Agents	Description
PLB to OPB Bridge Controller	PLB42OPB0 (master 0)
Direct Memory Access Controller	DMA2P40 (master 1)

2.2.3 OPB Arbiter Registers

.

Table 2-10. OPB Arbiter Registers

Mnemonic	Register Name	Address	Access	Page
OPBA0_PR	OPB 0 Arbiter Priority Register	0x4 EF60 0A00	R/W	132
OPBA0_CR	OPB 0 Arbiter Control Register	0x4 EF60 0A01	R/W	132

2.2.3.1 OPB Arbiter Priority Register (OPBAn_PR)

The OPBAn_PR assigns priorities to the OPB master IDs.

Figure 2-26. OPB Arbiter Priority Register (OPBAn_PR)

0:1	HPM	High Priority Master ID 00 Master ID 0of OPB master device connected to M0_request and OPB_M0Grant 01 Master ID of OPB master device connected to M1_request and OPB_M1Grant 10 Reserved 11 Reserved	
2:3	MHP	Medium High Priority master ID 00 Master ID of OPB master device connected to M0_request and OPB_M0Grant 01 Master ID of OPB master device connected to M1_request and OPB_M1Grant 10 Reserved 11 Reserved	
4:7		Reserved	

2.2.3.2 OPB Arbiter Control Register (OPBAn_CR)

The OPBAn_CR fields controls updating of the OPBAn_PR (described in *OPB Arbiter Priority Register (OPBAn_PR)* on page 132).

Figure 2-27. OPB Arbiter Control Register (OPBAn_CR)

0	DPE	Dynamic Priority Enable 0 Dynamic priority disabled 1 Dynamic priority enabled	
1	PEN	Park Enable 0 Park disabled 1 Park enabled	
2	PMN	Park on Master Not Last 0 Park on master last 1 Park on master not last	
3:4	PID	Parked Master ID 00 Master ID 0 of OPB master device connected to M0_request and OPB_M0Grant 01 Master ID 1 of OPB master device connected to M1_request and OPB_M1Grant 10 Reserved 11 Reserved	
5:7		Reserved	

2.3 Advanced High Performance Bus (PPC460EX/EXr only)

The following sections describe the AHB bus. This bus is provided only on the PPC460EX/EXr.

2.3.1 AHB Features

The advanced high performance bus features:

- A 32-bit address bus and a 32-bit data bus
- Issues all transfer types—IDLE, BUSY, SEQ and NONSEQ
- Supports all burst types for both read and write transactions
  - Single Transfer
  - Incrementing burst transfers (INCR4, INCR8 and INCR16)
  - Wrapping Burst transfers (WRAP4, WRAP8 and WRAP16)
  - Variable length Burst transfers (INCR)
- Supports all transfer sizes: byte, halfword, and fullword transfers
- Issues transfer responses—SPLIT, RETRY, OKAY, and ERROR
- Supports up to 200 MHz AHB clock

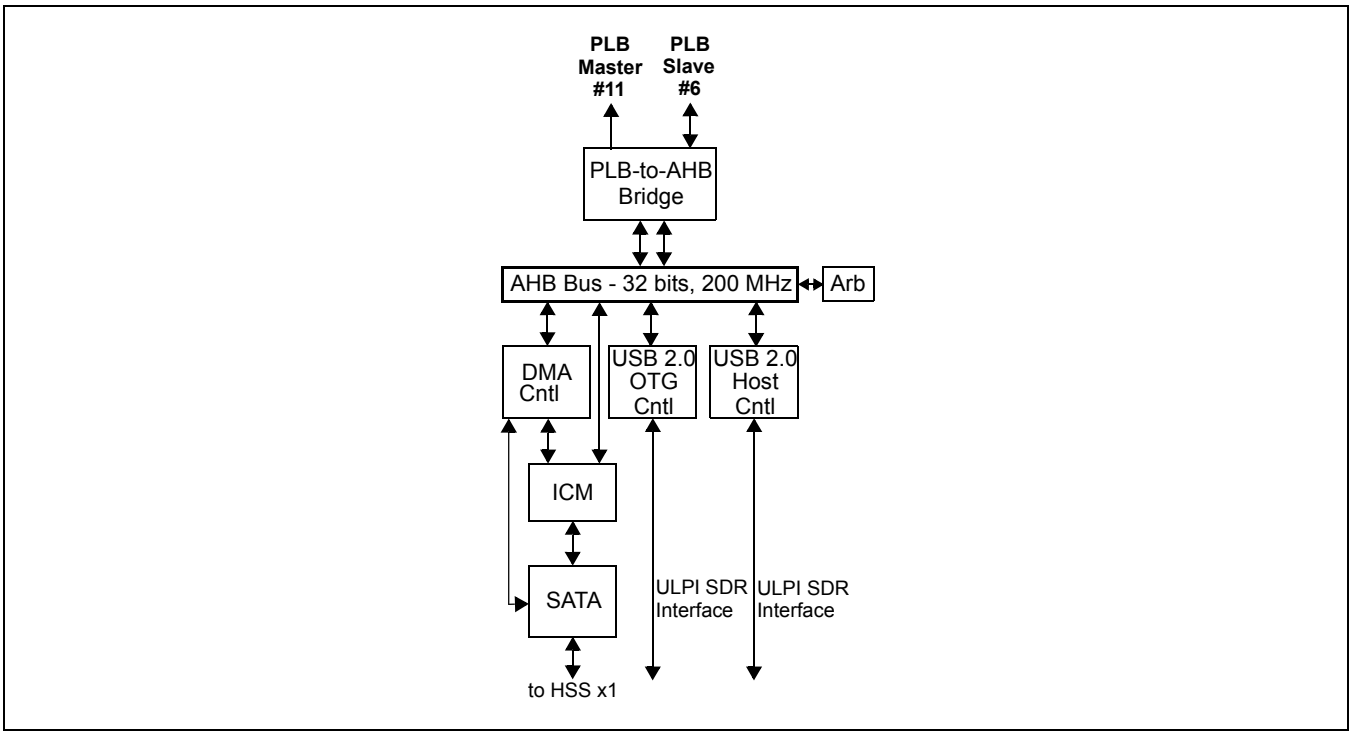
The AHB is a high performance bus used by three high-speed serial interfaces to connect to the PLB through a bidirectional bridge. The AHB Subsystem connects two USB 2.0 controllers, a Host Controller and an On-The-Go (OTG) Controller. Additionally, there is a Serial ATA controller connected the AHB bus through a DMA controller. The AHB Subsystem master and slave assignments for PPC460EX/EXr/GT are listed at *AHB Masters and Slaves Assignments* on page 136.

The bidirectional bridge between the PLB and AHB, PLB4XAHB, enables data transfers between PLB masters and AHB slaves, AHB masters and PLB slaves.

For more details on the AHB architecture, refer to the *AMBA Specification, Revision 2.0*.

2.3.2 AHB Subsystem Functional Block Diagram

Figure 2-28. AHB Subsystem Functional Block Diagram



2.3.3 AHB Subsystem Address Mapping

The registers for the AHB Subsystem can be accessed through PLB (through AHB) and DCR address spaces. The PLB address mapping is shown in *Table 2-11*, AHB address mapping is shown in *Table 2-12*, and the DCR address mapping is shown in *Table 2-13*. Most of the devices are accessed through the PLB as an AHB Peripheral. The exception is the PLB-to-AHB Bridge which is accessed through DCR.

2.3.3.1 AHB Subsystem PLB Address Mapping

The AHB slave devices occupy 512KB of PLB address space between 0x0000_0004_BFF8_0000 and 0x0000_0004_BFFF_FFFF.

Table 2-11. AHB Subsystem PLB Address Mapping

Device	Start Address	End Address	Size
USB2.0 OTG	0x0000 0004 BFF8 0000	0x0000 0004 BFFB FFFF	256KB
USB2.0 Host (OHCI)	0x0000 0004 BFFD 0000	0x0000 0004 BFFD 03FF	1KB
USB2.0 Host (EHCI)	0x0000 0004 BFFD 0400	0x0000 0004 BFFD 07FF	1KB
AHBDMA for SATA	0x0000 0004 BFFD 0800	0x0000 0004 BFFD 0BFF	1KB
SATA	0x0000 0004 BFFD 1000	0x0000 0004 BFFD 17FF	2KB
AHB Arbiter	0x0000 0004 BFFD 2000	0x0000 0004 BFFD 23FF	1KB

2.3.3.2 AHB Subsystem AHB Address Mapping

The entire 4GB of AHB address space is mapped to the PLB-to-AHB bridge except for the 512KB decoded for the other AHB slaves.

Table 2-12. AHB Subsystem AHB Address Mapping

Device	Start Address	End Address	Size[1]
PLB-to-AHB Bridge (Control and Status Registers (CSRs))	0x0000 0000	0x0000 00FF	256B
PLB-to-AHB Bridge (PLB Address Space)	0x0000 0100	0xBFF7 FFFF	~3GB
USB2.0 OTG	0xBFF8 0000	0xBFFB FFFF	256KB
USB2.0 Host (OHCI)	0xBFFD 0000	0xBFFD 03FF	1KB
USB2.0 Host (EHCI)	0xBFFD 0400	0xBFFD 07FF	1KB
AHBDMA for SATA	0xBFFD 0800	0xBFFD 0BFF	1KB
SATA	0xBFFD 1000	0xBFFD 17FF	2KB
AHB Arbiter	0xBFFD 2000	0xBFFD 23FF	1KB
PLB-to-AHB Bridge (PLB Address Space)	0xC000 0000	0xFFFF FFFF	1GB

Notes:
1. Each AHB slave has a minimum address decode space of 1KB.
2. For AHB-to-PLB transfers, bits 28:31 of the PLB UA Bus come from bits 6:3 of the AHB2PLB Bridge Control Register.

2.3.3.3 AHB Subsystem DCR Address Mapping

Table 2-13. AHB Subsystem DCR Address Mapping

Device	Start Address	End Address	Size
PLB-to-AHB Bridge	0x0A0	0x0AF	16W

2.3.3.4 AHB Master and Slave Assignment

The following table lists the AHB masters and slaves provided in PPC460EX/EXr.

Table 2-14. AHB Masters and Slaves Assignments

AHB Agent	AHB Masters and Slaves	
AHB Arbiter		Slave #0
PLB4XAHB Bridge	Master #1	Slave #1
SATA DMA	Master #2	Slave #2
USB OTG	Master #3	Slave #3
USB Host OHCI port	Master #4	Slave #4
USB Host EHCI port	Master #5	Slave #5
SATA Host (through ICM)		Slave #6

2.3.4 AHB Subsystem SDR Registers

Table 2-15 lists the SDR0 registers used to configure AHB Subsystem devices. The subsequent sections provide detailed information about the individual registers.

Table 2-15. AHB Subsystem SDR Registers

Mnemonic	Register	Address	Access	Page
SDR0_AHB_CFG	AHB Subsystem Configuration Register	0x0370	R/W	137
SDR0_USB2HOST_CFG	USB2 Host Configuration Register	0x0371	R/W	137
SDR0_USB2HOST_STS	USB2 Host Status Register	0x0372	R	138
SDR0_SATA_CFG	USB2 SATA Configuration Register	0x0373	R/W	138

2.3.4.1 AHB Subsystem Configuration Register (SDR0_AHB_CFG)

Figure 2-29. AHB Subsystem Configuration Register (SDR0_AHB_CFG)			
0:22		Reserved	
23	AHB_STP	AHB Clock Stop	Reset value = SDR0_SDSTP1[AHBSTOP]
24	A2P_INCR4	AHB to PLB INCR to INCR4 Enable	When set to 1, enables logic which forces AHB transactions with INCR burst type targeting the PLB4XAHB bridge to INCR4 burst type. This is used to avoid a scenario for the PLB4XAHB bridge which may cause it hang.
25	A2P_PROT2	AHB to PLB HPROT2 Disable	When set to 1, enables logic which forces HPROT2 low for AHB transactions with INCRX burst type targeting the PLB4XAHB bridge. This is used to avoid a scenario for the PLB4XAHB bridge which may cause it hang.
26	DMAC_S_BE	DMA Controller Slave Big Endian	SATA DMAC Slave
27	DMAC_M2_BE	DMA Controller Master2 Big Endian	SATA DMAC Master 2
28	DMAC_M1_BE	DMA Controller Master1 Big Endian	SATA DMAC Master 1
29	OTG_M_BE	USB2 OTG Master Big Endian	
30	OTG_S_BE	USB2 OTG Slave Big Endian	
31	AHB_BE	AHB Arbiter Big Endian	

2.3.4.2 USB2 Host Configuration Register (SDR0_USB2HOST_CFG)

Figure 2-30. USB2 Host Configuration Register (SDR0_USB2HOST_CFG)			
0:15		Reserved	
16:19	OHCI_HPROT	OHCI AHB HPROT Value	AHB HPROT is not driven by the OHCI. Reset value = 4b0000.
20:23	EHCI_HPROT	EHCI AHB HPROT Value	AHB HPROT is not driven by the EHCI. Reset value = 4b0000.
24:25		Reserved	
26:31	EHCI_FLADJ	EHCI Frame Length Adjust Value	Frame length adjust register: Can only set when HCHAULT bit in USBSTS set to1. Otherwise EHCI yields undefined results. It shouldn't be reprogrammed by system software unless default BIOS programmed value is wrong. 59488 plus this value is uSOF cycle time to generate an uSOF microframe length. Reset value = 6b100000.

2.3.4.3 USB2 Host Status Register (SDR0_USB2HOST_STS)

Figure 2-31. USB2 Host Status Register (SDR0_USB2HOST_STS)			
0:8		Reserved	
9	OHCI_SPD	OHCI USB Speed	Indicates the speed at which the USB is operating
10	OHCI_SPND	OHCI Suspend	Indicates that the PHY is being put into a Suspend state.
11	OHCI_GSPND	OHCI Global Suspend	Indicates that USB2 Host Controller is in Global Suspend state. This signal is asserted 5ms after the USB2 Host Controller enters the USB Suspend state. It remains asserted as long as the host controller remains in this state.
12	OHCI_RWE	OHCI Remote Wake Up Enabled	Indicates whether Remote Wake Up is enabled or not.
13	OHCI_RMTWKP	OHCI Remote Wake Up Status	Indicates status of Remote Wake Up.
14	OHCI_DRWE	OHCI Device Remote Wake Up Enable	Indicates that the USB2 Host Controller is enabled to treat connect and disconnect as wake up events. This will cause the USB2 Host Controller to move from the Global Suspend state to the Global Resume state.
15	OHCI_CCS	OHCI Current Connect Status	When high, Indicates that the port state machine is in a connected state. When low, the port state machine is in either a disconnected or powered-off state.
16:19		Reserved	
20:23	EHCI_LPSMC_STATE	EHCI LPSMC State	Indicates the state of the LPSMC module.
24:25		Reserved	
26:31	EHCI_USBSTS	EHCI USB Status	Indicates pending interrupts and various USB2 Host Controller statuses. These six bits reflect the value in the USBSTS[5:0] register.

2.3.4.4 SATA Configuration Register (SDR0_SATA_CFG)

Figure 2-32. SATA Configuration Register (SDR0_SATA_CFG)			
0:29		Reserved	
30	COMWAKE_DISABLE	Device COMWAKE Disable	When set to 1, disables the ability of a SATA device to wake up the PHY and SATA logic using COMWAKE transaction. Power management must be enabled in order for this bit to have an effect (PWRMGMT_DISABLE = 0).
31	PWRMGMT_DISABLE	Power Management Disable	When set to 1, disables the power management circuit.

2.3.5 Bidirectional Bridge

The bidirectional bridge is a high performance bridge that allows transactions from the AHB to the PLB and vice versa.

The bridge consists of two components:

1. AHB2PLB bridge—this bridge is a slave on the AHB interface and master on the PLB interface.
2. PLB2AHB bridge —this bridge is a slave on the PLB interface and master on the AHB interface.

The AHB2PLB bridge converts the transactions from 32-bit wide AHB master to 128-bit wide PLB transfers on the PLB bus, DCR transfers on 32-bit wide DCR bus, or to the bridge's internal Control and Status registers. The PLB2AHB bridge converts the transactions from 64/128-bit wide PLB masters to 32-bit wide AHB transactions on the AHB bus.

Each bridge uses three FIFOs:

The first one is WFIFO, which is a 128 bits wide by 4-deep synchronous dual-clock FIFO, for queueing the write data and byte enables from PLB. A buffer register is used to collect the write data from PLB before the write data is pushed into WFIFO. The byte enables are treated in the same way as the write data.

Another 4-deep synchronous dual-clock FIFO, AFIFO, is used to store the information such as the start address, read/write, transaction type, and the byte count for the transaction. PLB address and transfer qualifiers are latched into a buffer register first. The address bits are compared with the upper and lower address ranges in the min_ahb_addr_reg and max_ahb_addr_reg registers to determine if the transaction is for AHB. If it is, then based on the conditions such as space availability in FIFOs, the transaction type and PLB abort, a decision is made to either accept the current transaction or ask the PLB to rearbitrate the transaction. If the transaction is aborted, nothing is pushed into AFIFO.

RFIFO is also a four-deep synchronous dual-clock FIFO for queuing read data from AHB. A read buffer collects the read data from AHB, and appropriately address aligns it. When the address advances to the next 16-byte boundary, the read buffer's data is pushed into RFIFO. If the amount of data to be read does not warrant crossing a 16-byte boundary, the data is pushed anyway into the RFIFO, when the required amount of data is collected. When a read transfer is complete on the AHB side, PLB interface starts the PLB read data transfer to complete the PLB read transaction. It is to be emphasized that the RFIFO is not used as a buffer. It is used solely as a method of synchronizing between the AHB and PLB clock domains.

Since the AFIFO is 4 deep, PLB transactions can be queued, especially write transactions since writes are always posted. The following set of rules is used in deciding when to queue a PLB transaction in AFIFO:

- Queue single writes as long as WFIFO and AFIFO have space (not full).
- Queue a read transaction if AFIFO has space. Note that at a time only one read transaction is queued in the AFIFO.
- If WFIFO is not empty, no burst (variable- or fixed-length) or 8/16 word line write transaction is queued. A 4-word line write is queued if WFIFO is not full.

The bridge asserts rearbitration on PLB when a valid transaction cannot be accepted by the bridge due to unavailable resources.

During primary read transfers, when the bridge's read resources are free, the bridge responds with rearbitration on PLB when there are posted writes in the other direction of the bridge. The bridge waits for the transfer to become primary transfer and then asserts rearbitration on PLB based on the availability of bridge resources.

Write-errors on the AHB interface cause the SEAR and SESR registers to be updated (if they're not locked), the current AHB operation is terminated and the WFIFO is flushed. The PLB interface will then assert PLB interrupt. Read-errors on the AHB interface cause the information to be propagated to the PLB interface, which then asserts error condition on the PLB. The AHB interface checks for 1KB page boundary when the Guarded storage attribute is set, so the read and write bursts do not cross the 1KB page boundary.

If a write request comes in first and then a read request comes in, the AHB interface ensures that the write operation completes first before a read operation. If the write operation ends with an ERROR response, then the read to the same address gets the old data.

The AHB to PLB bridge is the slave on the AHB interface and master on the PLB interface. It allows a AHB master to use the PLB bus. The bridge PLB interface is 128 bits, but the bridge supports both 64- and 128-bit PLB slaves. The bridge's AHB interface is a 32-bit interface.

2.3.5.1 AHB Registers

The bridge Control and Status Registers (CSRs) are mapped in the DCR address space.The following sections describe the memory map of the bridge CSRs in the DCR space.

The reset value of all the registers in the following table is 0.

Table 2-16. AHB Registers

Mnemonic	Register	DCR Address	Access	Page
AHB_REV	Revision ID Register	0x00A0	Read only	140
AHB_SEARU	PLB2AHB System Error Upper Address Register	0x00A1	R/W	141
AHB_SEARL	PLB2AHB System Error Lower Address Register	0x00A2	R/W	141
AHB_SESR	PLB2AHB System Error Status Register	0x00A3	R/W	141
AHB_TOP	PLB2AHB Top Address Register	0x00A4	R/W	141
AHB_BOT	PLB2AHB Bottom Address Register	0x00A5	R/W	141
AHB_ATT	PLB2AHB Attribute Register	0x00A6	R/W	142
AHB_CR	AHB2PLB Control Register	0x00A7	R/W	142
AHB_ES	AHB2PLB Error Status Register	0x00A8	R/W	142
AHB_EA	AHB2PLB Error Address Register	0x00A9	R/W	142
AHB_IM	AHB2PLB Interrupt Mask Register	0x00AA	R/W	143

2.3.5.2 Revision ID Register (AHB_REV)

Figure 2-33. PLB2AHB Revision ID Register (AHB_REV)

31:0	REVID	32 bits Revision ID	

2.3.5.3 PLB2AHB System Error Upper Address Register (AHB_SEARU)

Figure 2-34. PLB2AHB System Error Upper Address Register (AHB_SEARU)			
31:4		Reserved	
3:0	AU	Least significant 4-bits of offending PLB_UABUS upon first error	Write to this register will clear the contents of the register

2.3.5.4 PLB2AHB System Error Lower Address Register (AHB_SEARL)

Figure 2-35. PLB2AHB System Error Lower Address Register (AHB_SEARL)			
31:0	AL	32-bit of the offending address upon the first error	

2.3.5.5 PLB2AHB System Error Status Register (AHB_SESTR)

Figure 2-36. PLB2AHB System Error Status Register (AHB_SESR)			
31	RdErr	First reported error was due to a read	Locked upon first error (either read or write)
30	WrErr	First reported error was due to a write	Locked upon first error (either read or write)
29:14	MIRQ	Master Interrupt Request received	
13:4		Reserved	
3:0	MIN	Master Number for the first error reported	

2.3.5.6 PLB2AHB Top/Bottom Address Register (AHB_TOP/AHB_BOT)

The AHB address range is defined by these two registers.

The recommended setting for both registers is 0x8000000E

Figure 2-37. PLB2AHB Top / Bottom Address Register (AHB_TOP/AHB_BOT)			
31	VAL	Valid bit	
30:8		Reserved	
7:0	TOP/BOT	Top/Bottom address 7:4 forms PLB_UABUS28:31 3:0 forms PLB_ABUS0:3	

2.3.5.7 PLB2AHB Attribute Register (AHB_ATT)

Figure 2-38. PLB2AHB Attribute Register (AHB_ATT)			
31:1		Reserved	
0	FNBUF	Force Non-buffered transfer on AHB When set, the PLB_TAttribute8 is ignored	

2.3.5.8 AHB2PLB Bridge Control Register (AHB_CR)

Figure 2-39. AHB2PLB Bridge Control Register (AHB_CR)			
31:8		Reserved	
7	PAPD	PLB Address Pipelining Disable	
6:3	PUOA	PLB Upper Order Address	
2:1	PRP	PLB Request Priority	Sets request priority on PLB: 11 - Highest ... 00 - Lowest
0	LOCK	PLB Lock Error	

2.3.5.9 AHB2PLB Error Status Register (AHB_ES)

Figure 2-40. AHB2PLB Error Status Register (AHB_ES)			
31:4		Reserved	
3	PIRQ	PLB MIRQ	
2	RdErr	PLB Read Transfer Error	
1	WrErr	PLB Write Transfer Error	
0	TIME	PLB Master Timeout Error	

These errors are combined to form a single interrupt to the UIC when it is enabled by the AHB_IM register

2.3.5.10 AHB2PLB Error Address Register (AHB_EA)

Figure 2-41. AHB2PLB Error Address Register (AHB_EA)			
31:0	ADD	PLB Address	

2.3.5.11 AHB2PLB Interrupt Mask Register (AHB_IM)

Figure 2-42. AHB2PLB Interrupt Mask Register (AHB_IM)			
31:4		Reserved	
3	PIRQ	PLB MIRQ Mask	
2	RdErr	PLB Read Transfer Error Mask	
1	WrErr	PLB Write Transfer Error Mask	
0	TIME	PLB Master Timeout Error Mask	

2.3.6 System Considerations

The following sections provide some system considerations for the user.

2.3.6.1 LOCK Transfers

The bridge supports LOCK transactions from initiating masters in both directions.

PLB to AHB—locked transactions from the PLB master are converted to LOCK transfers on the AHB bus.

AHB to PLB—the bridge's PLB master interface does not initiate LOCK transfers on PLB bus. However, when transferring the LOCK transaction from AHB bus, the bridge forces the highest priority.

In general, the bridge executes read and write transactions in the order received. However, when there is a LOCK transfer from an AHB master, any pending read transfers are aborted from other AHB masters that were previously given SPLIT, and any data in the Read Data FIFO is flushed and split response is given to the original AHB master. Subsequently, the LOCK transfer is allowed to complete.

When the master that received the split response comes back to complete the read (after the AHB bus is free), the read is treated as a fresh read, and the master is SPLIT once again to complete the read. At the time of receiving the locked transaction, all posted writes in the bridge are completed on the destination bus.

If ORDERING has been enabled in the bidirectional bridge and the fresh incoming transfer to AHB slave interface of the bridge is a LOCK transfer, then ordering is ignored by the bridge until the lock transfer completes on the AHB bus. This avoids deadlock conditions that can occur with ordering enabled, and also avoids fresh lock transfers initiated on the AHB bus.

The SPLIT response mechanism on the AHB bus gives adequate protection against the dead lock scenario in the bidirectional bridge. However, deadlock can occur when locked transfer types are used simultaneously by both the AHB master and PLB master. Therefore you should avoid configurations in which of both AHB master and PLB master can start LOCKed transaction at exactly the same time to prevent an irrecoverable deadlock condition.

2.3.6.2 Error Handling

Error Handling: PLB to AHB

If a write ERROR response occurs on the AHB side, the bridge terminates the AHB burst transaction immediately. If there are data remaining in WFIFO, WFIFO is flushed. SEAR and SESR registers are updated. The bridge asserts the appropriate Interrupt signal to the PLB master.

If a read ERROR response is received on the AHB bus, the bridge terminates the AHB burst transaction immediately. The bridge terminates the PLB burst transactions when the remaining data in Read Data FIFO is empty. The appropriate error and acknowledgement are asserted for the last PLB read transfer. SEAR and SESR registers are updated.

Error Handling: AHB to PLB

During a write, when PLB timeout error or PLB slave write error occurs on the PLB side:
Since writes are posted in the bridge, the bridge continues to finish the intended bufferable write on the PLB bus. The Error status registers and error address registers are updated. If interrupts are enabled, the bridge generates a non-maskable interrupt in response to the error received from a PLB slave device.

During a prefetchable read, or non-bufferable read and write:
The bridge asserts an ERROR response on the AHB bus for the particular address for which PLB slave gave a write error or read error or timeout error. The bridge updates the error address and error status registers.

If there is a continuation of the same transfer by AHB master after ERROR response, the bridge issues a RETRY for the subsequent transfer. Thus the slave interface responds with RETRY when an ERROR is encountered on the PLB bus on a prefetchable read or on a nonbufferable read/write, and the transfer is continued by the AHB master. This RETRY allows the master to initiate the transfer with newly formed burst for the remaining transfer, or for a fresh new transfer. During this RETRY response, the AHB slave interface does not record any transaction information internally and treats the next incoming transfer as a fresh transfer.

2.3.7 AHB Arbiter

The AHB arbiter allows only one bus master to have access to the bus at any given time. Each master can request control of the bus; however, which one is given a grant is determined by the programmable priority level of each master.

2.3.7.1 Arbitration Schemes

Masters can be masked from the arbitration scheme. Requests are masked according to protocol requirements, such as split-slave transactions or a programmed priority level of 0.

First Tier Arbitration – When two or more masters request the bus at the same time, the requesting master with the highest priority is granted the bus.

Second Tier Arbitration ("Fair Among Equals") – If two requesting masters have the same priority, then ownership is based on a "Fair-Among-Equals" algorithm. This algorithm compares each master at every clock cycle. When a master is granted access to the bus, the arbiter holds the grant for two clock cycles (providing hready is high). Then it checks which master is requesting access and re-arbitration occurs. There is also a possibility that some wait states occur before the bus is granted to the next master. For instance, one master can be granted ownership of the bus for several accesses before the arbiter grants the bus to the next master. A fixed-length burst could continue into its SEQ portion by taking advantage of this extra grant cycle, allowing the burst to complete. Arbitration is locked during the fixed-length burst until beat 7 (assuming INCR8) and hready is high to allow back-to-back transfers on the bus.

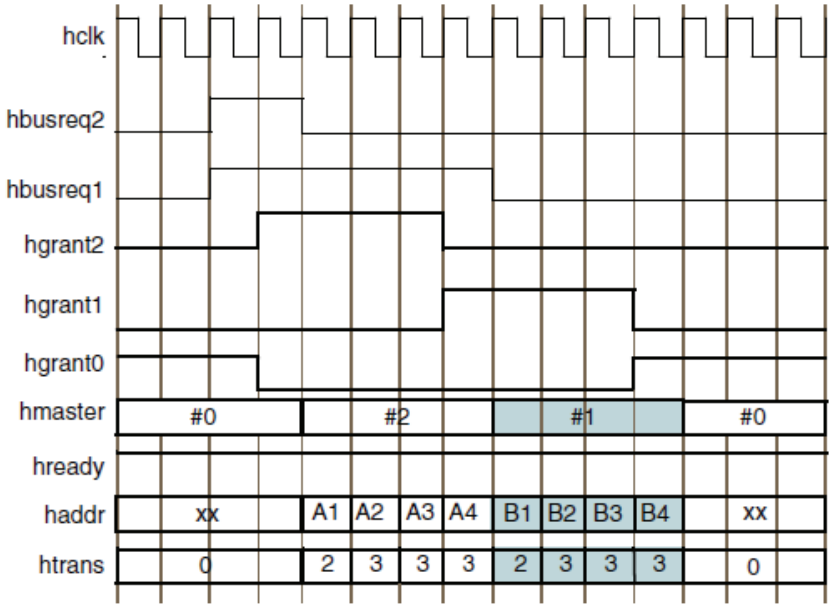
With this scheme, there is the slight possibility that a master can get "starved" from the bus, such as when there are three masters vying for access to the bus. Assuming there are no wait states, the bus grants ownership as follows: M1–M3–M2–M1. In this example, the arbiter does not have a way of stopping the updating of the Fair-Among-Equals arbitration when the result from the arbiter is not used—it still cycles through its Fair-Among-Equals algorithm. In this type of scenario, there is a possibility that a master (for example, M2) does not get granted

ownership if the number of clients and wait states is of a combination that the Fair-Among-Equals algorithm (which is running free) is not arbitrating because every time the arbiter is polled to produce the grant, the same master (for example, M1) surfaces again.

The granting of masters for fixed-length burst masters is not every cycle. Therefore, the fairness depends on the number of masters and wait states, and on each cycle calculating the next master. If the system is not ready, then another master is granted in the following cycle. The master that wins ownership is the master that is granted when the bus is ready. All masters will eventually be granted ownership.

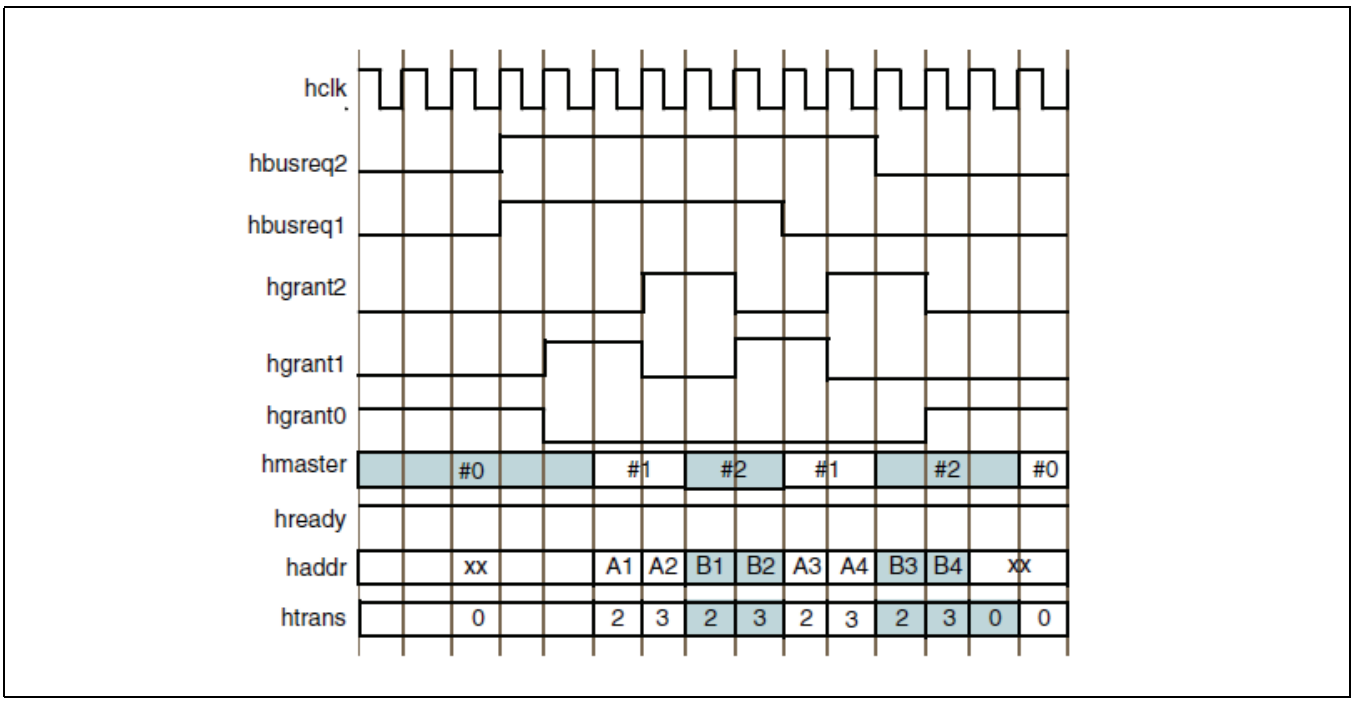
The following figure shows the timing for fixed-length bursts with multiple masters requesting ownership. When masters have different priorities, the bus ownership is highest down to lowest. When masters have equal priorities, the bus ownership is random.

Figure 2-43. Bursts With Specified Length



The following figure shows the timing that occurs when masters with undefined length bursts and the same priority level are early terminated by masters of equal or higher priority.

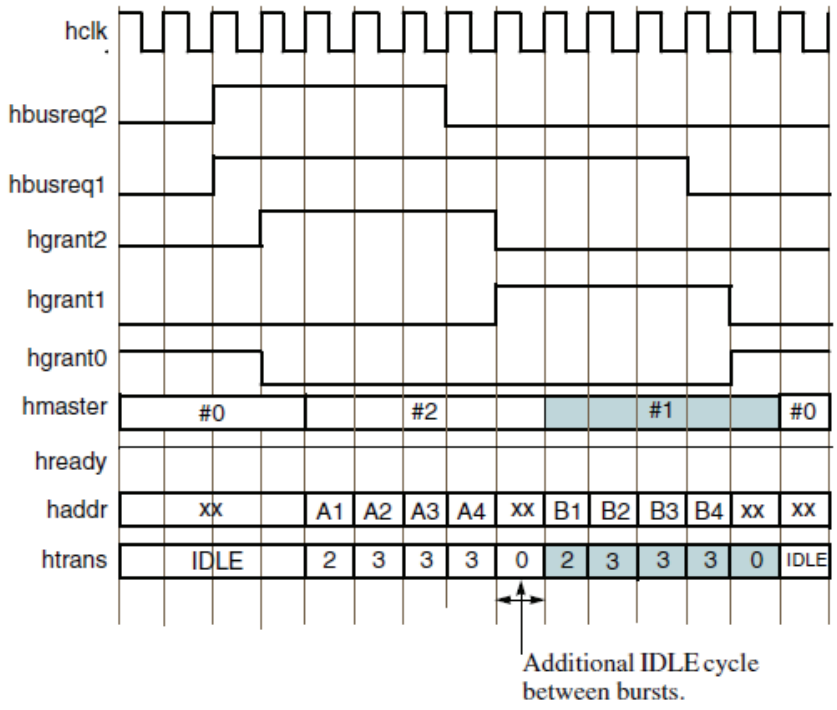
Figure 2-44. Bursts with Unspecified Length and Equal Priority Level



In the previous figure, there are two masters requesting ownership of the bus with the same priority. When a master is granted access to the bus, it is for a minimum of two clock cycles. The arbiter then selects one of the masters, and bus ownership can change. Burst transfers that are of an unspecified length may be early burst terminated by grants to requests from masters of equal priority or greater priority. For bursts of this type, changing grants is calculated at every cycle.

The following figure shows the timing for bursts of an unspecified length from masters with different priority levels. The highest priority wins ownership of the bus each time. There is an additional IDLE cycle between bursts, because the request line has to be held until the last transfer has started.

Figure 2-45. Bursts with Unspecified Length and Different Priority Level



2.3.7.2 Transfers

The arbiter supports many types of transfers, including split and locked transfers. If a split-capable slave is not able to complete a transfer as requested, it can issue a split response to its master. In this case, the arbiter may grant ownership of the bus to another requesting bus master through the normal method of arbitration. The master who received the split response will not be granted bus access until the slave indicates it is ready to resume the split transfer.

A master that has been split cannot begin another transfer on the bus until the original transfer has been completed because it is masked from arbitration. When a slave indicates to the arbiter that the split transfer can resume, the master is allowed to compete under the normal arbitration procedure. The master must complete the same transfer.

A slave may issue a retry response if it is not capable of completing a transfer. In this case, the arbiter will begin arbitration again to grant access to the bus, but masters with a lower priority than the current master will be masked out. This is in contrast to the split response where all other requesting masters may compete for access to the bus.

When a master attempts to access an unassigned address, the arbiter returns an error response.

For more details on the AHB architecture, refer to the *AMBA Specification, Revision 2.0*.

2.3.8 AHB Arbiter Registers

The following table lists the AHB arbiter registers.

Table 2-17. AHB Arbiter Registers

Mnemonic	Description	Address	Reset Value	Access	Page
Master Priority Level					
AHBARB0_PL1	Arbitration Priority Level for Master 1	0x4 BFFD 2000	0xF	R/W	148
AHBARB0_PL2	Arbitration Priority Level for Master 2	0x4 BFFD 2004	0xE	R/W	148
AHBARB0_PL3	Arbitration Priority Level for Master 3	0x4 BFFD 2008	0xD	R/W	148
AHBARB0_PL4	Arbitration Priority Level for Master 4	0x4 BFFD 200C	0xD	R/W	148
AHBARB0_PL5	Arbitration Priority Level for Master 5	0x4 BFFD 2010	0xD	R/W	148
Early Burst Termination					
AHBARB0_EBTCNT	Early Burst Termination Count	0x4 BFFD 203C	0	R/W	149
AHBARB0_EBTEN	Early Burst Termination Enable	0x4 BFFD 2040	0	R/W	149
AHBARB0_EBTSTS	Early Burst Termination Status	0x4 BFFD 2044	0	R/C	149
Default Master					
AHBARB0_DFT_MST	Default Master ID Number	0x4 BFFD 2048	1	R/W	149
AHBARB0_AHB_VERSION	Version ID	0x4 BFFD 2090	0x3230362A	R	149

2.3.8.1 AHB Arbitration Priority Master n Register (AHBARB0_PLn)

Each AHB master n has its own arbitration priority level register, AHBARB0_PLn. The priority level ranges from 0 to 15, with a priority 0 signifying that the master has been disabled. Highest priority is given to masters with priority 15, while masters with priority 1 have the lowest level of priority.

Figure 2-46. AHB Arbitration Priority Master n Register (AHBARB0_PLn)

31:4		Reserved	
3:0	PL	Arbitration priority for master n	

2.3.8.2 Early Burst Termination Feature

The Early Burst Termination feature is implemented through a programmable 10-bit counter in the arbiter. This counter can be enabled or disabled by writing to AHBARB0_EBTEN[EBT_EN]. If AHBARB0_EBTEN[EBT_EN] = 1 (is enabled) and a master assumes ownership of the bus, the counter assumes the value contained in the AHBARB0_EBTCNT[EBT_CNT]. The counter decrements on each clock tick during the transfer. If it reaches zero before the transfer completes, then the arbiter will terminate the master's ownership and commence arbitration between other competing masters. When there is an early burst termination, AHBARB0_EBTSTS[EBT_STS] is asserted and an interrupt is sent to the UIC2[27]. It is cleared whenever the register is read. As only one master has ownership of the bus at any given time, only one counter needs to be instantiated to implement the Early Burst Termination capability. The counter will be disabled by default, and will be cleared when reset.

2.3.8.3 Early Burst Termination Count Register (AHBARB0_EBTCNT)

Figure 2-47. Early Burst Termination Count Register (AHBARB0_EBTCNT)			
31:10		Reserved	
9:0	EBTCNT	Early burst termination count register Maximum number of cycles a transfer can take before being subject to an early burst termination.	

2.3.8.4 Early Burst Termination Enable Register (AHBARB0_EBTEN)

Figure 2-48. Early Burst Termination Enable Register (AHBARB0_EBTEN)			
31:1		Reserved	
0	EBTEN	Early burst termination enable.	

2.3.8.5 Early Burst Termination Status Register (AHBARB0_EBTSTS)

Figure 2-49. Early Burst Termination Status Register (AHBARB0_EBTSTS)			
31:1		Reserved	
0	EBTSTS	Early burst termination status Set when an Early Burst Termination takes place.	The register is cleared when read by the processor.

2.3.8.6 Default Master ID Number Register (AHBARB0_DFTMST)

If no master is requesting bus access, the AHB arbiter will grant the bus to the default master. The ID number of the designated default master can be programmed in the AHBARB0_DFTMST register. After a reset, the default master is the PLB-to-AHB bridge.

Figure 2-50. Default Master ID Number Register (AHBARB0_DFTMST)			
31:4		Reserved	
3:0	DFTMST	Default master ID number register The default master is the master that is granted by the bus when no master has requested ownership.	

2.3.8.7 Version ID Register (AHBARB0_VERSION)

Figure 2-51. Version ID Register (AHBARB0_VERSION)			
31:0	VERSION	ASCII value for each number in the version, followed by *.	For example, 32_30_36_2A represents the version 2.06*.

2.4 Device Control Register (DCR) Bus

The DCR bus provides a mechanism for the PPC460EX/EXr/GT to setup other on-chip facilities. For example, programmable resources in the DDR-SDRAM controller are configured for use with various memory devices according to their transfer characteristics and address assignments. DCR's are accessed through the use of the PowerPC **mfdcr** and **mtdcr** instructions.

The DCR bus also allows the PPC440 CPU to communicate with peripheral devices without using the PLB interface, thereby avoiding the impact to the primary system bus bandwidth, and without additional segmentation of the usable address map.

Part II. Processor

3. Programming Model

The programming model of the PPC460EX/EXr/GT corresponds to the programming model of the PPC440H6 CPU. The *PPC440H6 Processor User's Manual* describes how the following features and operations of the processor appear to programmers:

- Memory organization and addressing
- Registers
- Data types and alignment
- Byte ordering
- Instruction processing
- Branch processing
- Speculative accesses
- Privileged mode operation
- Synchronization
- Instruction set

3.1 Storage Addressing

As a 32-bit implementation of the Book-E Enhanced PowerPC Architecture, the PPC460EX/EXr/GT implements a uniform 32-bit effective address (EA) space. Effective addresses are expanded into virtual addresses and then translated to 36-bit (64 GB) real addresses by the memory management unit (see *Memory Management* on page 185 for more information on the translation process).

The PPC460EX/EXr/GT generates an effective address whenever it executes a storage access, branch, cache management, or translation look aside buffer (TLB) management instruction, or when it fetches the next sequential instruction.

Refer to the PPC460EX, PPC460EXr, or PPC460GT data sheets for detailed memory and DCR maps.

3.2 Registers

PPC460EX/EXr/GT registers are discussed in this section. Some of the frequently-used registers are described in detail. Other registers are covered in their respective topic chapters.

The PPC460EX/EXr/GT registers are grouped into categories: General Purpose Registers (GPRs), Special Purpose Registers (SPRs), Time Base Registers (TBRs), the Machine State Register (MSR), the Condition Register (CR), Device Control Registers (DCRs), and memory-mapped I/O registers (MMIO). Different instructions are used to access each category of registers.

The GPRs, SPRs, TBRs, the MSR, and the CR are covered in the *PPC440H6 Processor User's Manual.* The DCRs and MMIO registers are covered in this book.

For all registers with fields marked as reserved, the reserved fields should be written as 0 and read as undefined. That is, when writing to a register with a reserved field, write a 0 to the reserved field. When reading from a register with a reserved field, ignore that field.

Programming Note: A good coding practice is to perform the initial write to a register with reserved fields as described, and to perform all subsequent writes to the register using a read-modify-write strategy: read the register, use logical instructions to alter defined fields, leaving reserved fields unmodified, and write the register.

3.2.1 Device Control Registers

Device Control Registers (DCRs) are used to control various on-chip system functions such as the operation of on-chip buses, peripherals, and certain processor behaviors. The DCR access instructions are **mtdcr** (move-to-device control register) and **mfdcr** (move-from-device control register), which move data between GPRs and the DCRs.

Indirectly addressed DCRs are accessed using the SDR0 configuration registers SDR0_CFGADDR and SDR0_CFGDATA. In indirect addressing, the **mtdcr** and **mfdcr** instructions access a given SDR0 register by means of its address offset which is contained in the SDR0_CFGADDR register. The data for the specified register is moved to or moved from the SDR0_CFGDATA register. See *Table 3-1*.

All of the DCRs are listed in *Alphabetical Summary of Chip Control and Peripheral Function Registers* on page 1573.

3.2.1.1 System DCR Configuration Registers

All of the PPC460EX/EXr/GT system DCR registers (SDRs) are accessed indirectly using SDR0 Configuration Registers described in the following table.

Table 3-1. SDR0 Configuration Registers

Mnemonic	Register	Address	Access
SDR0_CFGADDR	System DCR Configuration Address Register	0x000E	R/W
SDR0_CFGDATA	System DCR Configuration Data Register	0x000F	R/W

Figure 3-1. SDR0 Configuration Address Register (SDR0_CFGADDR)

0:16		Reserved	
17:31	DCRA	15-bit SDR0 Register Offset Value	This value can range from 0x0000 to 0x7FFF. Its contents are used as the indirect DCR address for accessing an SDR0 register.

Figure 3-2. SDR0 Configuration Data Register (SDR0_CFGDATA)

0:31	DCRD	32-bit Data Value	This value can range from 0x00000000 to 0xFFFFFFFF. Its contents contains the value of the SDR0 register as indicated by the SDR0_CFGADDR register.

3.2.2 System DCRs (SDRs)

The system DCRs control miscellaneous system level functions and configurations. The SDR logic implements a standard DCR address space using indirect addressing.

Note: The specific page on which detailed register information is available can be located by finding the register name in the Index. The register name appears twice in the Index–once under "registers" and again under the letter with which the register mnemonic begins.

3.2.3 Memory-Mapped Input/Output Registers

Some registers associated with on-chip peripherals are memory-mapped input/output (MMIO) registers. Such registers are mapped into the system memory space and are accessed using load/store instructions that contain the register addresses.

3.3 Data Types and Alignment

Refer to the *PPC440H6 Processor User's Manual* for details.

3.4 Instruction Processing

Refer to the *PPC440H6 Processor User's Manual* for details.

4. FPU Programming Model

The PPC460EX/EXr/GT has a built-in super scalar FPU that supports both single- and double-precision operations, and offers single cycle through put on most instructions.

Features include:
- Five stages with 2 MFlops/MHz
- Hardware support for IEEE 754
- Single- and double-precision
- Single-cycle throughput on most instructions
- Thirty-two 64-bit floating point registers

The programming model of the PPC440 FPU describes how the following features and operations appear to programmers:

- Storage addressing (including storage operands, effective address calculation, and data storage addressing modes), starting on page 157
- Floating-point exceptions, starting on page 159
- Floating-point registers, starting on page 160
- Floating-point data formats, starting on page 163
- Floating-point execution models, starting on page 170
- Floating-point instructions, starting on page 173

The Book-E Enhanced PowerPC Architecture (referred to as Book-E) specifies that the floating-point unit (FPU) implements a floating-point system as defined in ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic* (referred to as IEEE 754), but the architecture requires software support to conform fully with the standard. IEEE 754 defines certain required "operations" (addition, subtraction, and so on); the term "floating-point operation" is used to refer to one of these required operations, or to the operation performed by one of the *Multiply-Add* or *Reciprocal Estimate* instructions. All floating-point operations conform to the IEEE standard, unless software sets the IEEE Mode (NI) bit to 1 in the Floating-Point Status and Control Register (FPSCR). When FPSCR[NI] = 1, floating-point operations do not necessarily conform to the IEEE standard.

Important: Before using the FPU, it must be enabled and configured in the processor MSR and CCR0 registers. Specifically, set MSR[FP] = 1 and CCR0[DAPUIB] = 0. Also, MSR[FE0, FE1] must be set as desired. Refer to the MSR and CCR0 registers in the *PPC440H6 Processor User's Manual* for details.

4.1 Storage Addressing

The PPC440 FPU accesses storage in the same uniform 32-bit (4GB) effective address (EA) space as the PPC440 processor. Effective addresses are expanded into virtual addresses and then translated to 33-bit (8GB) real addresses by the memory management unit (MMU) of the processor.

The PPC440 FPU generates an effective address whenever it executes a *Load/Store* instruction.

4.1.1 Storage Operands

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

The data storage operands accessed by the PPC440 FPU load/store instructions can be words (4 bytes, or 32 bits) or double words (8 bytes, or 64 bits). The address of a storage operand is the address of its first byte (that is, of its lowest-numbered byte). Byte ordering can be either big endian or little endian, as controlled by the endian (E) storage attribute.

Operand length is implicit for each scalar storage access instruction. The operand of such a scalar storage access instruction has a "natural" alignment boundary equal to the operand length. In other words, the "natural" address of an operand is an integral multiple of the operand length. A storage operand is said to be *aligned* if it is aligned at its natural boundary; otherwise, it is said to be *unaligned*.

Data storage operands for storage access instructions have the following characteristics.

Table 4-1. Data Operand Definitions

Storage Access Instruction Type	Operand Length	$A_{28:31}$ if aligned
Word	4 bytes	0bxx00
Doubleword	8 bytes	0bx000
Note: An "x" in an address bit position indicates that the bit can be 0 or 1 regardless of the state of other bits in the address.		

The alignment of the operand effective address of some storage access instructions can affect performance, and in some cases can cause an Alignment exception to occur. For such storage access instructions, the best performance is obtained when the storage operands are naturally aligned. *Table 4-2* summarizes the effects of alignment on those storage access instruction types for which such effects exist. If an instruction type is not shown in the table, then there are no alignment effects for that instruction type.

Table 4-2. Alignment Effects for Storage Access Instructions

Storage Access Instruction Type	Alignment Effect
FP Load/Store Word	Alignment exception if the storage crosses a 16-byte boundary ($EA_{28:31}$ = 0b1100); otherwise, no effect
FP Load/Store Doubleword	Alignment exception if the storage crosses 16-byte boundary ($EA_{28:31}$ > 0b1000); otherwise no effect

Instruction storage operands, on the other hand, are a word, and the effective addresses calculated by branch instructions are therefore always word-aligned.

4.1.2 Effective Address Calculation

For a storage access instruction, if the sum of the effective address and the operand length exceeds the maximum effective address of $2^{32} - 1$ (that is, the storage operand itself crosses the maximum address boundary), the result of the operation is undefined, as specified by the architecture. The processor performs the operation as if the storage operand wrapped around from the maximum effective address to effective address 0. Software, however, should not depend upon this behavior, so that can be ported to other implementations that do not handle such accesses in the same manner. Software should ensure that no data storage operands cross the maximum address boundary.

Note that because instructions are words, and because the effective addresses of instructions are always implicitly on word boundaries, an instruction storage operand cannot cross any word boundary, including the maximum address boundary.

Effective address arithmetic, which calculates the starting address for storage operands, wraps around from the maximum address to address 0, for all effective address computations except next sequential instruction fetching.

4.1.3 Data Storage Addressing Modes

The PPC440 FPU supports the following data storage addressing modes.

- Base + displacement (D-mode) addressing mode:

  The 16-bit D field is sign-extended to 32 bits and added to the contents of the GPR designated by RA, or to zero if RA = 0. The low-order 32 bits of the sum form the effective address of the data storage operand.

- Base + index (X-mode) addressing mode:

  The contents of the GPR designated by RB (or the value 0 for **lswi** and **stswi**) are added to the contents of the GPR designated by RA, or to zero if RA = 0; the low-order 32 bits of the sum form the effective address of the data storage operand.

4.2 Floating-Point Exceptions

Each floating-point exception, and each category of Invalid Operation Exception, is associated with an exception bit in the FPSCR. The following floating-point exceptions are detected by the processor; the associated FPSCR fields are listed with each exception and Invalid Operation exception category:

Table 4-3. Invalid Operation Exception Categories

Category	FPSCR Field
SNaN	VXSNAN
Infinity – Infinity	VXISI
Infinity ÷ Infinity	VXIDI
Zero ÷ Zero	VXZDZ
Infinity × Zero	VXIMZ
Invalid Compare	VXVC
Software Request	VXSOFT
Invalid Square Root	VXSQRT
Invalid Integer Convert	VXCVI

- Invalid Operation Exception (VX)
- Zero Divide Exception (ZX)
- Overflow Exception (OX)
- Underflow Exception (UX)
- Inexact Exception (XI)

Each floating-point exception also has a corresponding enable bit in the FPSCR. See *Floating-Point Status and Control Register Instructions* on page 180 for descriptions of these exception and enable bits, and *Floating Point Unit Interrupts and Exceptions* on page 269 for a detailed discussion of floating-point exceptions, including the effects of the FPSCR enable bits.

4.3 Floating-Point Registers

This section provides an overview of the register types implemented in the PPC440 FPU. Detailed descriptions of the floating-point registers are provided within the chapters covering the functions with which they are associated.

Certain bits in some registers are *reserved* and thus not necessarily implemented. For all registers with fields marked as reserved, these reserved fields should be written as 0 and read as *undefined*. The recommended coding practice is to perform the initial write to a register with reserved fields set to 0, and to perform all subsequent writes to the register using a read-modify-write strategy: read the register; use logical instructions to alter defined fields, leaving reserved fields unmodified; and write the register.

Each register is classified as being of a particular *type*, as characterized by the specific instructions used to read and write registers of that type. The registers contained within the PPC440 are defined by Book-E, except for Device Control Registers (DCRs) that are implementation-specific and unique to the PPC440 FPU.

4.3.1 Register Types

The PPC440 floating point unit provides three types of registers, Floating Point Registers (FPRs), the FPSCR, and DCRs. Each type is characterized by the instructions used to read and write the registers. The following subsections provide an overview of each register type and the instructions associated with them.

4.3.1.1 Floating-Point Registers (FPR0:31)

The PPC440 FPU provides 32 Floating-Point Registers (FPRs), each 64 bits wide. In any cycle, the FPR file can read the operands for a store instruction and an arithmetic instruction, or write the data from a load instruction and the result of an arithmetic instruction.

Figure 4-1. Floating-Point Registers (FPR0:31)

0:63	FPRD	Floating-Point Register data	

The FPRs are numbered FPR0:FPR31. The floating-point instruction formats provide 5-bit fields to specify the FPRs used as operands in the execution of the associated instructions.

Each FPR contains 64 bits that support the floating-point double format. All instructions that interpret the contents of an FPR as a floating-point value uses the floating-point double format for this interpretation.

The computational instructions, and the *Move* and *Select* instructions, operate on data located in FPRs and, with the exception of the *Compare* instructions, place the result value into a FPR and optionally place status information into the Condition Register (CR).

Load and store double instructions are provided that transfer 64 bits of data between storage and the FPRs with no conversion. *Load Single* instructions transfer and convert floating-point values in floating-point single format from storage to the same value in floating-point double format in the FPRs. Store single instructions are provided to transfer and convert floating-point values in floating-point double format from the FPRs to the same value in floating-point single format in storage.

Some floating-point instructions update the FPSCR and CR explicitly. Some of these instructions move data to and from an FPR to the FPSCR, or from the FPSCR to an FPR.

The computational instructions and the *Select* instruction accept values from the FPRs in double format. For single-precision arithmetic instructions, all input values must be representable in single format;. if not, the result placed into the target FPR, and the setting of status bits in the FPSCR are undefined.

4.3.1.2 Floating-Point Status and Control Register (FPSCR)

The FPSCR controls the handling of floating-point exceptions and records status resulting from the floating-point operations. See *Floating-Point Status and Control Register* on page 279 for a more detailed description of the FPSCR.

Figure 4-2. Floating-Point Status and Control Register (FPSCR)

0	FX	Floating-Point Exception Summary 0 No FPSCR exception bits changed from 0 to 1. 1 At least one FPSCR exception bit changed from 0 to 1.	All floating-point instructions, except **mtfsfi** and **mtfsf**, implicitly set this field to 1 if the instruction causes any floating-point exception bits in the FPSCR to change from 0 to 1. **mcrfs**, **mtfsfi**, **mtfsf**, **mtfsb0,** and **mtfsb1** can alter this field explicitly.
1	FEX	Floating-Point Enabled Exception Summary	The OR of all the floating-point exception fields masked by their respective enable fields. **mcrfs**, **mtfsfi**, **mtfsf**, **mtfsb0**, and **mtfsb1** cannot alter this field explicitly.
2	VX	Floating-Point Invalid Operation Exception Summary	The OR of all the Invalid Operation exception fields. **mcrfs**, **mtfsfi**, **mtfsf**, **mtfsb0**, and **mtfsb1** cannot alter this field explicitly.
3	OX	Floating-Point Overflow Exception 0 A Floating-Point Overflow exception did not occur. 1 A Floating-Point Overflow exception occurred.	See *Overflow Exception* on page 275
4	UX	Floating-Point Underflow Exception 0 A Floating-Point Underflow exception did not occur. 1 A Floating-Point Underflow exception occurred.	See *Underflow Exception* on page 276
5	ZX	Floating-Point Zero Divide Exception 0 A Floating-Point Zero Divide exception did not occur. 1 A Floating-Point Zero Divide exception occurred.	See *Zero Divide Exception* on page 275
6	IX	Floating-Point Inexact Exception 0 A Floating-Point Inexact exception did not occur. 1 A Floating-Point Inexact exception occurred.	This field is a sticky version of FPSCR[FI] The following rules describe how a given instruction sets this field. If the instruction affects FPSCR[FI], the new value of this field is obtained by ORing the old value of this field with the new value of FPSCR[FI]. If the instruction does not affect FPSCR[FI], the value of this field is unchanged.
7	VXSNAN	Floating-Point Invalid Operation Exception (SNaN) 0 A Floating-Point Invalid Operation exception (VXSNAN) did not occur. 1 A Floating-Point Invalid Operation exception (VXSNAN) occurred.	See *Invalid Operation Exception* on page 273
8	VXISI	Floating-Point Invalid Operation Exception ($\infty - \infty$) 0 A Floating-Point Invalid Operation exception (VXISI) did not occur. 1 A Floating-Point Invalid Operation exception (VXISI) occurred.	See *Invalid Operation Exception* on page 273
9	VXIDI	Floating-Point Invalid Operation Exception ($\infty \div \infty$) 0 A Floating-Point Invalid Operation exception (VXIDI) did not occur. 1 A Floating-Point Invalid Operation exception (VXIDI) occurred.	See *Invalid Operation Exception* on page 273

10	VXZDZ	Floating-Point Invalid Operation Exception (0 ÷ 0) 0 A Floating-Point Invalid Operation exception (VXZDZ) did not occur. 1 A Floating-Point Invalid Operation exception (VXZDZ) occurred.	See *Invalid Operation Exception* on page 273
11	VXIMZ	Floating-Point Invalid Operation Exception (∞ × 0) 0 A Floating-Point Invalid Operation exception (VXIMZ) did not occur. 1 A Floating-Point Invalid Operation exception (VXIMZ) occurred.	See *Invalid Operation Exception* on page 273
12	VXVC	Floating-Point Invalid Operation Exception (Invalid Compare) 0 A Floating-Point Invalid Operation exception (VXVC) did not occur. 1 A Floating-Point Invalid Operation exception (VXVC) occurred.	See *Invalid Operation Exception* on page 273
13	FR	Floating-Point Fraction Rounded	The last *Arithmetic* or *Rounding and Conversion* instruction either produced an inexact result during rounding or caused a disabled Overflow Exception. See *Rounding* on page 169. This bit is not sticky.
14	FI	Floating-Point Fraction Inexact	The last *Arithmetic* or *Rounding and Conversion* instruction either produced an inexact result during rounding or caused a disabled Overflow Exception. See *Rounding* on page 169. This bit is not sticky. See the definition of FPSCR[XX] regarding the relationship between FPSCR[FI] and FPSCR[XX].
15	FPRF	Floating-Point Result Flag (FPRF)	
16	FL	Floating-Point Less Than or Negative	
17	FG	Floating-Point Greater Than or Positive	
18	FE	Floating-Point Equal to Zero	
19	FU	Floating-Point Unordered or NaN	
20		Reserved	
21	VXSOFT	Floating-Point Invalid Operation Exception (Software Request) 0 A Floating-Point Invalid Operation exception (Software Request) did not occur. 1 A Floating-Point Invalid Operation exception (Software Request) occurred.	See *Invalid Operation Exception* on page 273
22	VXSQRT	Floating-Point Invalid Operation Exception (Invalid Square Root) 0 A Floating-Point Invalid Operation exception (Invalid Square Root) did not occur. 1 A Floating-Point Invalid Operation exception (Invalid Square Root) occurred.	See *Invalid Operation Exception* on page 273
23	VXCVI	Floating-Point Invalid Operation Exception (Invalid Integer Convert) 0 A Floating-Point Invalid Operation exception (Invalid Integer Convert) did not occur. 1 A Floating-Point Invalid Operation exception (Invalid Integer Convert) occurred.	See *Invalid Operation Exception* on page 273
24	VE	Floating-Point Invalid Operation Exception Enabled 0 Floating-Point Invalid Operation exceptions are disabled. 1 Floating-Point Invalid Operation exceptions are enabled.	

25	OE	Floating-Point Overflow Exception Enable 0 Floating-Point Overflow exceptions are disabled. 1 Floating-Point Overflow exceptions are enabled.	
26	UE	Floating-Point Underflow Exception Enable 0 Floating-Point Underflow exceptions are disabled. 1 Floating-Point Underflow exceptions are enabled.	
27	ZE	Floating-Point Zero Divide Exception Enable 0 Floating-Point Zero Divide exceptions are disabled. 1 Floating-Point Zero Divide exceptions are enabled.	
28	XE	Floating-Point Inexact Exception Enable 0 Floating-Point Inexact exceptions are disabled. 1 Floating-Point Inexact exceptions are enabled.	
29	NI	Floating-Point Non-IEEE Mode 0 Non-IEEE mode is disabled 1 Non-IEEE mode is enabled.	If FPSCR[NI = 1, the remaining FPSCR bits may have meanings other than those given in this document, and the results of floating-point operations need not conform to the IEEE standard. If the IEEE-conforming result of a floating-point operation would be a denormalized number, the result of that operation is 0 (with the same sign as the denormalized number) if FPSCR[NI] = 1. The behavior when FPSCR[NI] = 1 can vary from one implementation to another
30:31	RN	Floating-Point Rounding Control 00 Round to nearest 01 Round toward zero 10 Round toward +Infinity 11 Round toward –Infinity	See *Rounding* on page 169.

Programming Note: Setting FPSCR[NI] = 1 is intended to permit results to be approximate and to cause performance to be more predictable and less data-dependent than when FPSCR[NI] = 0. For example, in non-IEEE mode, 0 is returned instead of a denormalized number, and non-IEEE mode may return a large number instead of an infinity. In non-IEEE mode, an implementation should provide a means for ensuring that all results are produced without software assistance (that is, without causing an Enabled exception type Program interrupt or a Floating-Point Unimplemented Instruction exception type Program interrupt, and without invoking an "emulation assist." See *Floating Point Unit Interrupts and Exceptions* on page 269 The means may be controlled by one or more other FPSCR bits (recall that the other FPSCR bits have implementation-dependent meanings when FPSCR[NI] = 1).

4.4 Floating-Point Data Formats

Floating-point values are represented in two binary fixed-length formats. Single-precision values are represented in the 32-bit single format. Double-precision values are represented in the 64-bit double format. The single format can be used for data in storage, but cannot be stored in the FPRs. The double format can be used for data in storage and for data in the FPRs. When a floating-point value is loaded from storage using a *Load Single* instruction, it is converted to double format and placed in the target FPR. Conversely, a floating-point value stored from an FPR into storage using a Store Single instruction is converted to single format before being placed in storage.

The lengths of the exponent and the fraction fields differ between these two formats. The structure of the single and double formats are shown in *Table 4-4* and *Table 4-5*, respectively.

Table 4-4. Floating-Point Single Format

S	EXP	Fraction	
0	1	9	31

Table 4-5. Floating-Point Double Format

S	EXP	Fraction	
0	1	12	63

Values in floating-point format are composed of three fields:

Table 4-6. Format Fields

Field	Description
S	Sign Bit
EXP	Exponent + bias
FRACTION	Fraction

If only a portion of a floating-point data item in storage is accessed, such as with a load or store instruction for a byte or half word (or word in the case of floating-point double format), the value affected depends on whether the PowerPC Embedded system is operating with big endian or little endian byte ordering.

4.4.1 Value Representation

Representation of numeric values in the floating-point formats consists of a sign bit (S), a biased exponent (EXP), and the fraction portion (FRACTION) of the significand. The significand consists of a leading implied bit concatenated on the right with the FRACTION. This leading implied bit is 1 for normalized numbers and 0 for denormalized numbers and is located in the unit bit position (that is, the first bit to the left of the binary point). Values representable within the two floating-point formats can be specified by the parameters listed in *Table 4-7*.

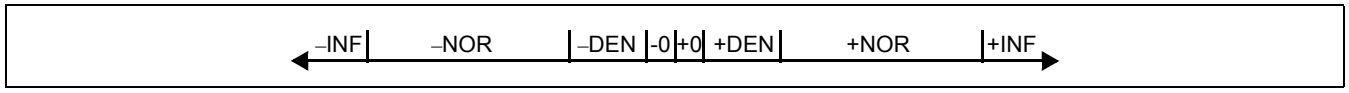
Table 4-7. IEEE 754 Floating-Point Fields

	Single	Double
Exponent Bias	+127	+1023
Maximum Exponent	+127	+1023
Minimum Exponent	–126	–1022
Field Widths (Bits)		
Sign	1	1
Exponent	8	11
Fraction	23	52
Significand	24	53

The FPRs support the floating-point double format only.

The numeric and nonnumeric values representable within each of the two supported formats are approximations to the real numbers and include the normalized numbers, denormalized numbers, and zero values. The nonnumeric values representable are the infinities and the Not a Numbers (NaNs). The infinities are adjoined to the real numbers, but are not numbers themselves, and the standard rules of arithmetic do not hold when they are used in an operation. They are related to the real numbers by order alone. It is possible, however. to define restricted operations among numbers and infinities. The relative location on the real number line for each of the defined entities is shown in *Figure 4-3*.

Figure 4-3. Approximation to Real Numbers



The NaNs are not related to the numeric values or infinities by order or value, but are encodings used to convey diagnostic information such as the representation of uninitialized variables.

The following is a description of the different floating-point values defined in the architecture:

4.4.2 Binary Floating-Point Numbers

Machine-representable values used as approximations to real numbers. Three categories of numbers are supported: normalized numbers, denormalized numbers, and zero values.

4.4.2.1 Normalized Numbers

Normalized numbers (±NOR) have an unbiased exponent value in the range:

- –126 to 127 in single format
- –1022 to 1023 in double format

They are values in which the implied unit bit is 1. Normalized numbers are interpreted as follows:

$$NOR = (-1)^{s} \times 2^{E} \times (1.fraction)$$

where s is the sign, E is the unbiased exponent, and 1.fraction is the significand, which is composed of a leading unit bit (implied bit) and a fraction part.

The ranges covered by the magnitude (M) of a normalized floating-point number are approximately equal to:

- Single Format:

  $1.2 \times 10^{-38} \leq M \leq 3.4 \times 10^{38}$

- Double Format:

  $2.2 \times 10^{-308} \leq M \leq 1.8 \times 10^{308}$

4.4.2.2 Denormalized Numbers

Denormalized numbers (±DEN) are values that have a biased exponent value of zero and a nonzero fraction value. They are nonzero numbers smaller in magnitude than the representable normalized numbers. They are values in which the implied unit bit is 0. Denormalized numbers are interpreted as follows:

$$DEN = (-1)^s \times 2^{Emin} \times (0.fraction)$$

where Emin is the minimum representable exponent value (–126 for single-precision, –1022 for double-precision).

4.4.2.3 Zero Values

Zero values (±0) have a biased exponent value of zero and a fraction value of zero. Zeros can have a positive or negative sign. The sign of zero is ignored by comparison operations; comparison treats +0 as equal to –0).

4.4.3 Infinities

Infinities (±∞)are values that have the maximum biased exponent value:

- 255 in single format
- 2047 in double format

and a zero fraction value. They are used to approximate values greater in magnitude than the maximum normalized value.

Infinity arithmetic is defined as the limiting case of real arithmetic, with restricted operations defined among numbers and infinities. Infinities and the real numbers can be related by ordering in the affine sense:

$$-\infty < \text{every finite number} < +\infty$$

Arithmetic on infinities is always exact and does not signal any exception, except when an exception occurs due to the invalid operations as described in *Invalid Operation Exception* on page 273.

4.4.3.1 Not a Numbers

Not a Numbers (NaNs) are values that have the maximum biased exponent value and a nonzero fraction value. The sign bit is ignored, that is, NaNs are neither positive nor negative. If the high-order bit of the fraction field is 0, the NaN is a Signalling NaN (SNaN); otherwise it is a Quiet NaN (QNaN).

Signaling NaNs are used to signal exceptions when they appear as operands of computational instructions.

Quiet NaNs are used to represent the results of certain invalid operations, such as invalid arithmetic operations on infinities or on NaNs, when Invalid Operation Exception is disabled (FPSCR[VE] = 0). Quiet NaNs propagate through all floating-point instructions except **fcmpo**, **frsp**, and **fctiw**. Quiet NaNs do not signal exceptions, except

for ordered comparison and conversion to integer operations. Specific encodings in QNaNs can thus be preserved through a sequence of floating-point operations, and used to convey diagnostic information to help identify results from invalid operations.

When a QNaN is the result of a floating-point operation because one of the operands is a NaN or because a QNaN was generated due to a disabled Invalid Operation exception, the following rule is applied to determine the NaN with the high-order fraction bit set to 1 that is to be stored as the result.

```
if FPR(FRA) is a NaN
then FPR(FRT) ← FPR(FRA)
else if FPR(FRB) is a NaN
then if instruction is frsp
        then FPR(FRT) ← FPR(FRB)0:34 || 29 0
        else FPR(FRT) ← FPR(FRB)
else if FPR(FRC) is a NaN
        then FPR(FRT) ← FPR(FRC)
        else if generated QNaN
                then FPR(FRT) ← generated QNaN
```

If the operand specified by FRA is a NaN, that NaN is stored as the result. Otherwise, if the operand specified by FRB is a NaN (if the instruction specifies an FRB operand), that NaN is stored as the result, with the low-order 29 bits of the result set to 0 if the instruction is **frsp**. Otherwise, if the operand specified by FRC is a NaN (if the instruction specifies an FRC operand), that NaN is stored as the result. Otherwise, if a QNaN was generated due to a disabled Invalid Operation Exception, that QNaN is stored as the result. If a QNaN is to be generated as a result, the QNaN generated has a sign bit of 0, an exponent field of all 1s, and a high-order fraction bit of 1 with all other fraction bits 0. Any instruction that generates a QNaN as the result of a disabled Invalid Operation must generate this QNaN (that is, 0x7FF8000000000000).

A double-precision NaN is representable in single format if and only if the low-order 29 bits of the double-precision NaNs fraction are zero.

4.4.4 Sign of Result

The following rules govern the sign of the result of an arithmetic, rounding, or conversion operation, when the operation does not yield an exception. They apply even when the operands or results are zeros or infinities.

- The sign of the result of an add operation is the sign of the operand having the larger absolute value. If both operands have the same sign, the sign of the result of an add operation is the same as the sign of the operands. The sign of the result of the subtract operation $x - y$ is the same as the sign of the result of the add operation $x + (-y)$.

When the sum of two operands with opposite sign, or the difference of two operands with the same sign, is exactly zero, the sign of the result is positive in all rounding modes except Round toward -Infinity, in which mode the sign is negative.

- The sign of the result of a multiply or divide operation is the Exclusive OR of the signs of the operands.
- The sign of the result of a *Square Root* or **frsqrte** instruction is always positive, except that the square root of –0 is –0 and the reciprocal square root of –0 is –Infinity.
- The sign of the result of an **frsp**[**.**], or **fctiw** operation is the sign of the operand being converted.

For the *Multiply-Add* instructions, the preceding rules are applied first to the multiply operation and then to the add or subtract operation (one of the inputs to the add or subtract operation is the result of the multiply operation).

4.4.5 Normalization and Denormalization

The intermediate result of an arithmetic or **frsp** instruction may require normalization and/or denormalization. Normalization and denormalization do not affect the sign of the result.

When an arithmetic or **frsp** instruction produces an intermediate result, consisting of a sign bit, an exponent, and a nonzero significand with a 0 leading bit, it is not a normalized number and must be normalized before it is stored.

A number is normalized by shifting its significand left while decrementing its exponent by 1 for each bit shifted, until the leading significand bit becomes 1. The G bit and the R bit (see *Execution Model for IEEE Operations* on page 171) participate in the shift with zeros shifted into the Round bit. The exponent is regarded as if its range were unlimited.

After normalization, or if normalization was not required, the intermediate result may have a nonzero significand and an exponent value that is less than the minimum value that can be represented in the format specified for the result. In this case, the intermediate result is said to be "Tiny" and the stored result is determined by the rules described in *Underflow Exception* on page 276. These rules may require denormalization.

A number is denormalized by shifting its significand right while incrementing its exponent by 1 for each bit shifted, until the exponent is equal to the format's minimum value. If any significant bits are lost in this shifting process, "Loss of Accuracy" has occurred (see *Underflow Exception* on page 276) and an Underflow Exception is signaled.

4.4.6 Data Handling and Precision

Instructions are defined to move floating-point data between the FPRs and storage. For double format data, the data are not altered during the move. For single format data, a format conversion from single to double is performed when loading from storage into an FPR. A format conversion from double to single is performed when storing from an FPR to storage. The *Load/Store* instructions do not cause floating-point exceptions.

All computational, *Move*, and **fsel** instructions use the floating-point double format.

Floating-point single-precision values are obtained with the following types of instruction.

- Load Floating-Point Single

  This form of instruction accesses a single-precision operand in single format in storage, converts it to double format, and loads it into an FPR. No floating-point exceptions are caused by these instructions.

- Round to Floating-Point Single-Precision

  The **frsp** instruction rounds a double-precision operand to single-precision, checking the exponent for single-precision range and handling any exceptions according to respective enable bits, and places that operand into an FPR as a double-precision operand. For results produced by single-precision arithmetic instructions, single-precision loads, and other instances of the **frsp** instruction, this operation does not alter the value.

Programming Note: The **frsp** instruction enables value conversion from double-precision to single-precision with appropriate exception checking and rounding. This instruction should be used to convert double-precision floating-point values (produced by double-precision load and arithmetic instructions) to single-precision values before storing them into single format storage elements or using them as operands for single-precision arithmetic instructions. Values produced by single-precision load and arithmetic instructions are already single-precision values and can be stored directly into single format storage elements, or used directly as operands for single-precision arithmetic instructions, without preceding the store, or the arithmetic instruction, by an **frsp** instruction.

- Single-Precision Arithmetic Instructions

  This form of instruction takes operands from the FPRs in double format, performs the operation as if it produced an intermediate result having infinite precision and unbounded exponent range, and then coerces this

intermediate result to fit in single format. Status bits in the FPSCR are set to reflect the single-precision result. The result is then converted to double format and placed into an FPR. The result lies in the range supported by the single format.

All input values must be representable in single format. If they are not, the result placed into the target FPR, and the setting of status bits in the FPSCR, are undefined.

- Store Floating-Point Single

  This form of instruction converts a double-precision operand to single format and stores that operand into storage. No floating-point exceptions are caused by these instructions. (The value being stored is effectively assumed to be the result of an instruction of one of the preceding three types.)

When the result of a *Load Floating-Point Single*, **frsp**, or single-precision arithmetic instruction is stored in an FPR, the low-order 29 fraction bits are zero.

Programming Note: A single-precision value can be used in double-precision arithmetic operations. The reverse is true only if the double-precision value is representable in single format.

4.4.7 Rounding

Rounding applies to operations that have numeric operands (operands that are not infinities or NaNs). Rounding the intermediate result of such operations may cause an Overflow Exception, an Underflow Exception, or an Inexact Exception. The following description assumes that the operations cause no exceptions and that the result is numeric. See *Value Representation* on page 164 and *Floating Point Unit Interrupts and Exceptions* on page 269 for the cases not covered here.

Execution Model for IEEE Operations on page 171 provides a detailed explanation of rounding.

The *Arithmetic* and *Rounding and Conversion* instructions produce intermediate results that can be regarded as having infinite precision and unbounded exponent range. Such intermediate results are normalized or denormalized if required, then rounded to the target format. The final result is then placed into the target FPR in double format or in integer format, depending on the instruction.

The *Arithmetic* and *Rounding and Conversion* instructions, which round intermediate results, set FPSCR[FR, FI]. If the fraction was incremented during rounding, FPSCR[FR] = 1; otherwise, FPSCR[FR] = 0. If the rounded result is inexact, FPSCR[FI] = 1; otherwise, FPSCR[FI] = 0.

The *Estimate* instructions set FPSCR[FR, FI] to undefined values. The remaining floating-point instructions do not alter FPSCR[FR, FI].

FPSCR[RN] specifies one of four programmable rounding modes.

Let z be the intermediate arithmetic result or the operand of a convert operation. If z can be represented exactly in the target format, then the result in all rounding modes is z as represented in the target format. If z cannot be represented exactly in the target format, let z1 and z2 bound z as the next larger and next smaller numbers representable in the target format. Then, z1 or z2 can be used to approximate the result in the target format.

Figure 4-4 shows the relation of z, z1, and z2 in this case. The following rules specify the rounding in the four modes. The abbreviation lsb means least-significant bit.

Figure 4-4. Selection of z1 and z2

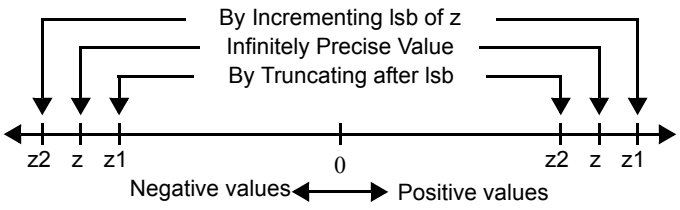


Table 4-8 describes the rounding modes.

Table 4-8. Rounding Modes

FPSCR[RN]	Rounding Mode	Description
00	Round to Nearest	Choose the value that is closest to z, either z1 or z2. In case of a tie, choose the one that is even (the lsb is 0).
01	Round toward Zero	Choose the smaller in magnitude (z1 or z2).
10	Round toward +Infinity	Choose z1.
11	Round toward –Infinity	Choose z2.

4.5 Floating-Point Execution Models

All implementations of this architecture must provide the equivalent of the following execution models to ensure that identical results are obtained.

Special rules are provided in the definition of the computational instructions for the infinities, denormalized numbers and NaNs. The material in the remainder of this section applies to instructions that have numeric operands and a numeric result (i.e., operands and result that are not infinities or NaNs), and that cause no exceptions. See *Value Representation* on page 164 and *Floating Point Unit Interrupts and Exceptions* on page 269 for the cases not covered here.

Although the double format specifies an 11-bit exponent, exponent arithmetic makes use of two additional bits to avoid potential transient overflow conditions. One extra bit is required when denormalized double-precision numbers are prenormalized. The second bit is required to permit the computation of the adjusted exponent value in the following cases when the corresponding exception enable bit is 1:

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

The IEEE standard includes 32-bit and 64-bit arithmetic. The standard requires that single-precision arithmetic be provided for single-precision operands. The standard permits double-precision floating-point operations to have either (or both) single-precision or double-precision operands, but states that single-precision floating-point operations should not accept double-precision operands. Book-E follows these guidelines: double-precision arithmetic instructions can have operands of either or both precisions, while single-precision arithmetic instructions require all operands to be single-precision. Double-precision arithmetic instructions produce double-precision values, while single-precision arithmetic instructions produce single-precision values.

For arithmetic instructions, conversions from double-precision to single-precision must be done explicitly by software, while conversions from single-precision to double-precision are done implicitly.

4.5.1 Execution Model for IEEE Operations

The following description uses 64-bit arithmetic as an example. 32-bit arithmetic is similar except that the FRACTION is a 23-bit field, and the single-precision Guard, Round, and Sticky bits (described in this section) are logically adjacent to the 23-bit FRACTION field.

IEEE-conforming significand arithmetic is considered to be performed with a floating-point accumulator having the following format, where bits 0:55 comprise the significand of the intermediate result.

Table 4-9. IEEE 64-bit Execution Model

S	C	L	FRACTION	G	R	X
		0	1 52			55

The S bit is the sign bit.

The C bit is the carry bit, which captures the carry out of the significand.

The L bit is the leading unit bit of the significand, which receives the implicit bit from the operand.

The FRACTION is a 52-bit field that accepts the fraction of the operand.

The Guard (G), Round (R), and Sticky (X) bits are extensions to the low-order bits of the accumulator. The G and R bits are required for post-normalization of the result. The G, R, and X bits are required during rounding to determine if the intermediate result is equally near the two nearest representable values. The X bit serves as an extension to the G and R bits by representing the logical OR of all bits that may appear to the low-order side of the R bit, due either to shifting the accumulator right or to other generation of low-order result bits. The G and R bits participate in the left shifts with zeros being shifted into the R bit. *Table 4-10* shows the significance of the G, R, and X bits with respect to the intermediate result (IR), the representable number next lower in magnitude (NL), and the representable number next higher in magnitude (NH).

Table 4-10. Interpretation of the G, R, and X Bits

G	R	X	Interpretation
0	0	0	IR is exact
0	0	1	IR closer to NL
0	1	0	
0	1	1	
1	0	0	IR midway between NL and NH
1	0	1	IR closer to NH
1	1	0	
1	1	1	

After normalization, the intermediate result is rounded, using the rounding mode specified by FPSCR[RN]. If rounding results in a carry into C, the significand is shifted right one position and the exponent incremented by one. This yields an inexact result and possibly also exponent overflow. Fraction bits to the left of the bit position used for rounding are stored into the FPR and low-order bit positions, if any, are set to zero.

Four user-selectable rounding modes are provided through FPSCR[RN] as described in *Rounding* on page 169. For rounding, the conceptual Guard, Round, and Sticky bits are defined in terms of accumulator bits. *Table 4-11* shows the positions of the Guard, Round, and Sticky bits for double-precision and single-precision floating-point numbers in the IEEE execution model.

Table 4-11. Location of the Guard, Round, and Sticky Bits in the IEEE Execution Model

Format	Guard	Round	Sticky
Double Single	G bit 24	R bit 25	X bit OR of 26:52, G, R, X

Rounding can be treated as though the significand were shifted right, if required, until the least significant bit to be retained is in the low-order bit position of the FRACTION. If any of the Guard, Round, or Sticky bits is nonzero, then the result is inexact.

z1 and z2, as defined in *Rounding* on page 169, can be used to approximate the result in the target format when one of the following rules is used.

- Round to Nearest
  - Guard bit = 0

    The result is truncated. (Result exact (GRX = 000) or closest to next lower value in magnitude (GRX = 001, 010, or 011))
  - Guard bit = 1

    Depends on Round and Sticky bits:
    - Case a

      If the Round or Sticky bit is 1 (inclusive), the result is incremented. (Result closest to next higher value in magnitude (GRX = 101, 110, or 111))
    - Case b

      If the Round and Sticky bits are 0 (result midway between closest representable values), then if the low-order bit of the result is 1 the result is incremented. Otherwise (the low-order bit of the result is 0) the result is truncated (this is the case of a tie rounded to even).
- Round toward Zero

  Choose the smaller in magnitude of z1 or z2. If the Guard, Round, or Sticky bit is nonzero, the result is inexact.
- Round toward +Infinity

  Choose z1.
- Round toward –Infinity

  Choose z2.

Where the result is to have fewer than 53 bits of precision because the instruction is a *Floating Round to Single-Precision* or single-precision arithmetic instruction, the intermediate result is either normalized or placed in correct denormalized form before being rounded.

4.5.2 Execution Model for Multiply-Add Type Instructions

The PPC440 FPU provides a special form of instruction that performs up to three operations in one instruction (a multiplication, an addition, and a negation). With this added capability comes the special ability to produce a more exact intermediate result as input to the rounder. 32-bit arithmetic is similar except that the FRACTION field is smaller.

Multiply-add significand arithmetic is considered to be performed with a floating-point accumulator having the following format, where bits 0:106 comprise the significand of the intermediate result.

Table 4-12. Multiply-Add 64-bit Execution Model

S	C	L	FRACTION	X'
		0	1 ... 105	106

The first part of the operation is a multiplication. The multiplication has two 53-bit significands as inputs, which are assumed to be prenormalized, and produces a result conforming to the above model. If there is a carry out of the significand (into the C bit), then the significand is shifted right one position, shifting the L bit (leading unit bit) into the most significant bit of the FRACTION and shifting the C bit (carry out) into the L bit. All 106 bits (L bit, the FRACTION) of the product take part in the add operation. If the exponents of the two inputs to the adder are not equal, the significand of the operand with the smaller exponent is aligned (shifted) to the right by an amount that is added to that exponent to make it equal to the other input's exponent. Zeros are shifted into the left of the significand as it is aligned and bits shifted out of bit 105 of the significand are ORed into the X' bit. The add operation also produces a result conforming to the above model with the X' bit taking part in the add operation.

The result of the addition is then normalized, with all bits of the addition result, except the X' bit, participating in the shift. The normalized result serves as the intermediate result that is input to the rounder.

For rounding, the conceptual Guard, Round, and Sticky bits are defined in terms of accumulator bits. *Table 4-13* shows the positions of the Guard, Round, and Sticky bits for double-precision and single-precision floating-point numbers in the multiply-add execution model.

Table 4-13. Location of Guard, Round, and Sticky Bits in the Multiply-Add Execution Model

Format	Guard	Round	Sticky
Double	53	54	OR of 55:105, X'
Single	24	25	OR of 26:105, X'

The rules for rounding the intermediate result are the same as those given in *Execution Model for IEEE Operations* on page 171.

If the instruction is *Floating Negative Multiply-Add* or *Floating Negative Multiply-Subtract*, the final result is negated.

4.6 Floating-Point Instructions

Primary opcode 63 is used for the double-precision arithmetic instructions and miscellaneous instructions, such as the *Floating-Point Status and Control Register Manipulation* instructions. Primary opcode 59 is used for the single-precision arithmetic instructions.

The single-precision instructions for which there is a corresponding double-precision instruction have the same format and extended opcode as the corresponding double-precision instruction.

Instructions are provided to perform arithmetic, rounding, conversion, comparison, and other operations in floating-point registers; to move floating-point data between storage and these registers; and to manipulate the FPSCR explicitly.

These instructions are divided into two categories.

- Computational instructions

  The computational instructions are those that perform addition, subtraction, multiplication, division, extracting the square root, rounding, conversion, comparison, and combinations of these operations. These instructions provide the floating-point operations. They place status information into the FPSCR. They are the instructions described in *Floating-Point Arithmetic Instructions* on page 178, *Floating-Point Rounding and Conversion Instructions* on page 179, and *Floating-Point Compare Instructions* on page 179.

- Noncomputational instructions

  The noncomputational instructions that perform loads and stores, move the contents of a floating-point register to another floating-point register possibly altering the sign, manipulate the FPSCR explicitly, and select a value from one of two floating-point registers based on the value in a third floating-point register. These operations are not considered floating-point operations. With the exception of the instructions that manipulate the FPSCR explicitly, they do not alter the FPSCR. Those instructions are described in *Floating-Point Status and Control Register Instructions* on page 180.

A floating-point number consists of a signed exponent and a signed significand. The quantity expressed by this number is the product of the significand and the number $2^{exponent}$. Encodings are provided in the data format to represent finite numeric values, ±infinity, and values that are "not a number" (NaN). Operations involving infinities produce results following traditional mathematical conventions. NaNs have no mathematical interpretation, but their encoding supports a variable diagnostic information field. NaNs may be used to indicate such things as uninitialized variables, and can be produced by certain invalid operations.

One class of exceptions that occur during floating-point instruction execution is unique to floating-point operations: the Floating-point exception. Bits set in the FPSCR indicate floating-point exceptions. They can cause an Enabled exception type Program interrupt to be taken, precisely or imprecisely, if the proper control bits are set.

4.6.1 Instructions By Category

The floating-point instructions can be classified into computational and noncomputational categories. The computational instructions include those that perform arithmetic operations or conversions on operands. Noncomputational instructions perform loads/stores and moves (with possible sign changes), or select data. Additionally, some noncomputational instructions can write directly to the FPSCR. All instructions executed in the Load/Store Pipeline are noncomputational, while most executed in the Arithmetic pipe are computational.

All floating-point operands are stored internally in Double Precision Format. Arithmetic operations specified as Single, require that the internal data is representable as Single (that is, having an unbiased exponent between –126 and 127 and a significand accurately representable in 24 bits). If the data cannot be represented in this way, the results stored in FPR, and the status bits set in FPSCR and CR (as appropriate), are undefined.

For consistency, to reduce the likelihood of causing a serious malfunction due to user error, and to enable random testing, single-precision operations are performed on double-precision operands. For all cases except for **fdivs**, the operation is performed as if it were double-precision; the result is then rounded to single-precision. For **fdivs**, the appropriate number of iterations are performed to accomplish a single-precision result (potentially with early out); the quotient is then properly rounded.

In all cases, result exceptions (overflow, underflow, and inexact) are detected and reported based on the result, not on the source operands. Default (masked exception) results are the same as for the single-precision instructions. In the case of masked overflow or underflow exceptions, the least significant 11 bits of the adjusted true exponent are returned.

The results of all single-precision operations are rounded to Single Precision. These results are stored in Double-Precision format, but are restricted to Single-Precision range (exponent and fraction). All status bits are set based upon the single-precision result.

4.6.2 Load and Store Instructions

The PPC440 FPU instruction set includes instructions to load from memory to an FPR, and to store from an FPR to memory.

For load instructions, the function of the LSC is to receive data from the 16 byte bus from the PPC460EX/EXr/GT and present it to the FPRs. Data received from PPC460EX/EXr/GT could be single or double precision, and in the big or little endian formats. Also, the data received is word aligned. Data to the FPR must be in the big endian, double precision format.

For store instructions one operand from the FPR, or a bypass path, is received. Data is to be word aligned on the output bus, not all cases can be handled with a throughput of one per cycle. FPR data (or bypassed data) for single precision stores that needs to be denormalized to fit in the single precision format requires multiple cycles. Also data for double precision stores may need to be normalized if the original data source was a single precision denormalized number. There are two basic forms of load instruction: single-precision and double-precision. Because the FPRs support only floating-point double format, single-precision *Load Floating-Point* instructions convert single-precision data to double format prior to loading the operand into the target FPR. The conversion and loading steps are as follows.

Let $WORD_{0:31}$ be the floating-point single-precision operand accessed from storage.

Normalized Operand
if $WORD_{1:8} > 0$ and $WORD_{1:8} < 255$ then
$FPR(FRT)_{0:1} \leftarrow WORD_{0:1}$
$FPR(FRT)_{2} \leftarrow \neg WORD_{1}$
$FPR(FRT)_{3} \leftarrow \neg WORD_{1}$
$FPR(FRT)_{4} \leftarrow \neg WORD_{1}$
$FPR(FRT)_{5:63} \leftarrow WORD_{2:31} \,||\, {}^{29}0$
Denormalized Operand
if $WORD_{1:8} = 0$ and $WORD_{9:31} \neq 0$ then
$sign \leftarrow WORD_{0}$
$exp \leftarrow -126$
$frac_{0:52} \leftarrow 0b0 \,||\, WORD_{9:31} \,||\, {}^{29}0$
normalize the operand
do while $frac_{0} = 0$
$frac \leftarrow frac_{1:52} \,||\, 0b0$
$exp \leftarrow exp - 1$
$FPR(FRT)_{0} \leftarrow sign$
$FPR(FRT)_{1:11} \leftarrow exp + 1023$
$FPR(FRT)_{12:63} \leftarrow frac_{1:52}$

Zero / Infinity / NaN
if $WORD_{1:8}$ = 255 or $WORD_{1:31}$ = 0 then
$FPR(FRT)_{0:1} \leftarrow WORD_{0:1}$
$FPR(FRT)_{2} \leftarrow WORD_{1}$
$FPR(FRT)_{3} \leftarrow WORD_{1}$
$FPR(FRT)_{4} \leftarrow WORD_{1}$
$FPR(FRT)_{5:63} \leftarrow WORD_{2:31} \,||\, {}^{29}0$

For double-precision *Load Floating-Point* instructions no conversion is required, as the data from storage are copied directly into the FPR.

Some of the *Floating-Point Load* instructions update GPR(RA) the effective address. For these forms, if RA $\neq$ 0, the effective address is placed into GPR(RA) and the storage element (byte, half word, word, or double word) addressed by EA is loaded into FPR(RT). If RA=0, the instruction form is invalid.

Floating-Point Load storage accesses cause Data Storage exceptions if the program is not allowed to read the storage location. *FLoating-Point Load* storage accesses cause Data TLB Error exceptions if the program attempts to access storage that is unavailable.

Note: RA and RB denote GPRs, while FRT denotes an FPR.

Both big endian and little endian byte orderings are supported.

Table 4-14. Floating-Point Load Instructions

Mnemonic	Operands	Instruction	Page
lfd	FRT, D(RA)	Load Floating-Point Double	1550
lfdu	FRT, D(RA)	Load Floating-Point Double with Update	1551
lfdux	FRT, RA, R	Load Floating-Point Double with Update Indexed	1552
lfdx	FRT, RA, R	Load Floating-Point Double Indexed	1553
lfs	FRT, D(RA)	Load Floating-Point Single	1554
lfsu	FRT, D(RA)	Load Floating-Point Single with Update	1555
lfsux	FRT, RA, RB	Load Floating-Point Single with Update Indexed	1556
lfsx	FRT, RA, RB	Load Floating-Point Single Indexed	1557

4.6.3 Floating-Point Store Instructions

There are three basic forms of store instruction: single-precision, double-precision, and integer. The integer form is provided by the **stfiwx** instruction, described on page 1568. Because the FPRs support only floating-point double format for floating-point data, single-precision *Store Floating-Point* instructions convert double-precision data to single format before storing the operand in storage. The conversion steps are as follows.

Let $WORD_{0:31}$ be the word in storage written to.

No Denormalization Required (includes Zero / Infinity / NaN)
if $FPR(FRS)_{1:11}$ > 896 or $FPR(FRS)_{1:63}$ = 0 then
$WORD_{0:1} \leftarrow FPR(FRS)_{0:1}$
$WORD_{2:31} \leftarrow FPR(FRS)_{5:34}$

Denormalization Required
if $874 \le FRS_{1:11} \le 896$ then
sign $\leftarrow FPR(FRS)_0$
exp $\leftarrow FPR(FRS)_{1:11} - 1023$
frac $\leftarrow$ 0b1 || $FPR(FRS)_{12:63}$
denormalize operand
do while exp < –126
frac $\leftarrow$ 0b0 || $frac_{0:62}$
exp $\leftarrow$ exp + 1
$WORD_0 \leftarrow$ sign
$WORD_{1:8} \leftarrow$ 0x00
$WORD_{9:31} \leftarrow frac_{1:23}$
else WORD $\leftarrow$ undefined

Notice that if the value to be stored by a single-precision *Store Floating-Point* instruction is larger in magnitude than the maximum number representable in single format, the first case above ("No Denormalization Required") applies. The result stored in WORD is then a well-defined value, but is not numerically equal to the value in the source register. The result of a single-precision *Load Floating-Point* from WORD will not compare equal to the contents of the original source register.

For double-precision *Store Floating-Point* instructions and for the *Store Floating-Point as Integer Word* instruction no conversion is required, as the data from the FPR are copied directly into storage.

Some of the *Floating-Point Store* instructions update GPR(RA) with the effective address. For these forms, if RA≠0, the effective address is placed into GPR(RA).

Floating-Point Store storage accesses will cause a Data Storage interrupt if the program is not allowed to write to the storage location. *Integer Store* storage accesses will cause a Data TLB Error interrupt if the program attempts to access storage that is unavailable.

Note: RA and RB denote GPRs, while FRS denotes an FPR.

Both big endian and little endian byte orderings are supported.

Table 4-15. Floating-Point Store Instructions

Mnemonic	Operands	Instruction	Page
stfd	FRS, D(RA)	Store Floating-Point Double	1564
stfdu	FRS, D(RA)	Store Floating-Point Double with Update	1565
stfdux	FRS, RA, RB	Store Floating-Point Double with Update Indexed	1566
stfdx	FRS, RA, RB	Store Floating-Point Double Indexed	1567
stfiwx	FRS, RA, RB	Store Floating-Point as Integer Word Indexed	1568
stfs	FRS, D(RA)	Store Floating-Point Single	1569
stfsu	FRS, D(RA)	Store Floating-Point Single with Update	1570
stfsux	FRS, RA, RB	Store Floating-Point Single with Update Indexed	1571
stfsx	FRS, RA, RB	Store Floating-Point Single Indexed	1572

4.6.4 Floating-Point Move Instructions

These instructions copy data from one floating-point register to another, altering the sign bit (bit 0) as described in the instruction descriptions in *Floating Point Instruction Set* on page 1517 for **fneg**, **fabs**, and **fnabs**. These instructions treat NaNs just like any other kind of value (for example, the sign bit of an NaN may be altered by **fneg**, **fabs**, and **fnabs**). These instructions do not alter the FSPCR.

Table 4-16. Floating-Point Move Instructions

Mnemonic	Operands	Instruction	Page
fabs	FRT, FRB	Floating Absolute Value	1521
fmr	FRT, FRB	Floating Move Register	1532
fnabs	FRT, FRB	Floating Negative Absolute Value	1537
fneg	FRT, FRB	Floating Negate	1538

4.6.5 Floating-Point Arithmetic Instructions

These instructions perform elementary arithmetic operations.

Table 4-17. Floating-Point Elementary Arithmetic Instructions

Mnemonic	Operands	Instruction	Page
fadd	FRT, FRA, FRB	Floating Add	1522
fadds	FRT, FRA, FRB	Floating Add Single	1523
fdiv	FRT, FRA, FRB	Floating Divide	1528
fdivs	FRT, FRA, FRB	Floating Divide Single	1529
fmul	FRT, FRA, FRB	Floating Multiply	1535
fmuls	FRT, FRA, FRB	Floating Multiply Single	1536
fres	FRT, FRB	Floating Reciprocal Estimate Single	1543
frsqrte	FRT, FRB	Floating Reciprocal Square Root Estimate	1545
fsub	FRT, FRA, FRB	Floating Subtract	1548
fsubs	FRT, FRA, FRB	Floating Subtract Single	1549

4.6.5.1 Floating-Point Multiply-Add Instructions

These instructions combine a multiply and an add operation without an intermediate rounding operation. The fraction part of the intermediate product is 106 bits wide (L bit, FRACTION), and all 106 bits take part in the add/subtract portion of the instruction.

FPSCR bits are set as follows.

- Overflow, Underflow, and Inexact Exception bits, the FR and FI bits, and the FPRF field are set based on the final result of the operation, and not on the result of the multiplication.

- Invalid Operation exception bits are set as if the multiplication and the addition were performed using two separate instructions (**fmul[s]**, followed by **fadd[s]** or **fsub[s]**. That is, multiplication of infinity by 0 or of anything by an SNaN, and addition of an SNaN, cause the corresponding exception bits to be set.

Table 4-18. Floating-Point Multiply-Add Instructions

Mnemonic	Operands	Instruction	Page
fmadd	FRT, FRA, FRB, FRC	Floating Multiply-Add	1530
fmadds	FRT, FRA, FRB, FRC	Floating Multiply-Add Single	1531
fmsub	FRT, FRA, FRB, FRC	Floating Multiply-Subtract	1533
fmsubs	FRT, FRA, FRB, FRC	Floating Multiply-Subtract Single	1534
fnmadd	FRT, FRA, FRB, FRC	Floating Negative Multiply-Add	1539
fnmadds	FRT, FRA, FRB, FRC	Floating Negative Multiply-Add Single	1540
fnmsub	FRT, FRA, FRB, FRC	Floating Negative Multiply-Subtract	1541
fnmsubs	FRT, FRA, FRB, FRC	Floating Negative Multiply-Subtract Single	1542

4.6.6 Floating-Point Rounding and Conversion Instructions

Examples of uses of these instructions to perform various conversions can be found in Appendix X, "Floating-Point Conversions", on page XREF TBD.

Table 4-19. Floating-Point Rounding and Conversion Instructions

Mnemonic	Operand	Instruction	Page
fctiw	FRT, FRB	Floating Convert To Integer Word	1526
fctiwz	FRT, FRB	Floating Convert To Integer Word and round to Zero	1527
frsp	FRT, FRB	Floating Round to Single-Precision	1544

4.6.7 Floating-Point Compare Instructions

The floating-point *Compare* instructions compare the contents of two floating-point registers. Comparison ignores the sign of zero (+0 is treated as equal to −0). The comparison result can be ordered or unordered.

The comparison sets one bit in the designated CR field to 1 and the other three bits to 0. FPSCR[FPCC] is set in the same way.

The CR field and FPSCR[FPCC] are set as follows.

Table 4-20. Comparison Sets

Bit	Name	Description
0	FL	(FRA) < (FRB)
1	FG	(FRA) > (FRB)
2	FE	(FRA) = (FRB)
3	FU	(FRA) ? (FRB) (unordered)

Table 4-21. Floating-Point Compare and Select Instructions

Mnemonic	Operands	Instruction	Page
fcmpo	BF, FRA, FRB	Floating Compare Ordered	1524
fcmpu	BF, FRA, FRB	Floating Compare Unordered	1525
fsel	FRT, FRA, FRB, FRC	Floating Select	1547

4.6.8 Floating-Point Status and Control Register Instructions

Every *Floating-Point Status and Control Register* instruction synchronizes the effects of all floating-point instructions executed by a given processor. Executing a *Floating-Point Status and Control Register* instruction ensures that all floating-point instructions previously initiated by the given processor have completed before the *Floating-Point Status and Control Register* instruction is initiated, and that no subsequent floating-point instructions are initiated by the given processor until the *Floating-Point Status and Control Register* instruction has completed. In particular:

- All exceptions that will be caused by the previously initiated instructions are recorded in the FPSCR before the *Floating-Point Status and Control Register* instruction is initiated.
- All invocations of the Enabled exception type Program interrupt that will be caused by the previously initiated instructions have occurred before the *Floating-Point Status and Control Register* instruction is initiated.
- No subsequent floating-point instruction that depends on or alters the settings of any FPSCR bits is initiated until the *Floating-Point Status and Control Register* instruction has completed.

Floating-Point Load and *Floating-Point Store* instructions are not affected.

Table 4-22. Floating-Point Status and Control Register Instructions

Mnemonic	Operands	Instruction	Page
mcrfs		Move To Condition Register From FPSCR	1558
mffs	FRT	Move From FPSCR	1559
mtfsb0	BT	Move To FPSCR Bit 0	1560
mtfsb1	BT	Move To FPSCR Bit 1	1561
mtfsf	FLM, FRB	Move To FPSCR Fields	1562
mtfsfi	BF,U	Move To FPSCR Field Immediate	1563

5. Cache Operations

The PPC460EX/EXr/GT incorporates two internal caches, a 32-KB instruction cache and a 32-KB data cache. Instructions and data can be accessed in the caches much faster than in main memory.

The instruction cache unit (ICU) controls instruction accesses to main memory and stores frequently used instructions to reduce the overhead of instruction transfers between the instruction pipeline and external memory. Using the instruction cache minimizes access latency for frequently executed instructions.

The data cache unit (DCU) controls data accesses to main memory and stores frequently used data to reduce the overhead of data transfers between the GPRs and external memory. Using the data cache minimizes access latency for frequently used data.

Refer to the *PPC440H6 Processor User's Manual* for details.

6. Memory Management

The PPC460EX/EXr/GT memory management unit (MMU) performs address translation and protection functions. With appropriate system software, the MMU supports:

- Translation of effective addresses to real addresses
- Independent enabling of instruction and data address translation and protection
- Page-level access control using the translation mechanism
- Software control of page replacement strategy
- Additional virtual-mode control of protection using zones
- Real-mode write protection

Refer to the *PPC440H6 Processor User's Manual* for details.

7. L2 Cache/SRAM Controller

The L2 cache controller (L2C0), which is between the PPC440 processor and the PLB, provides L2 cache or on-chip memory capability. The L2C0 attaches to the processor's three PLB master interfaces, three master ports on the PLB arbiter, and to a PLB slave port.

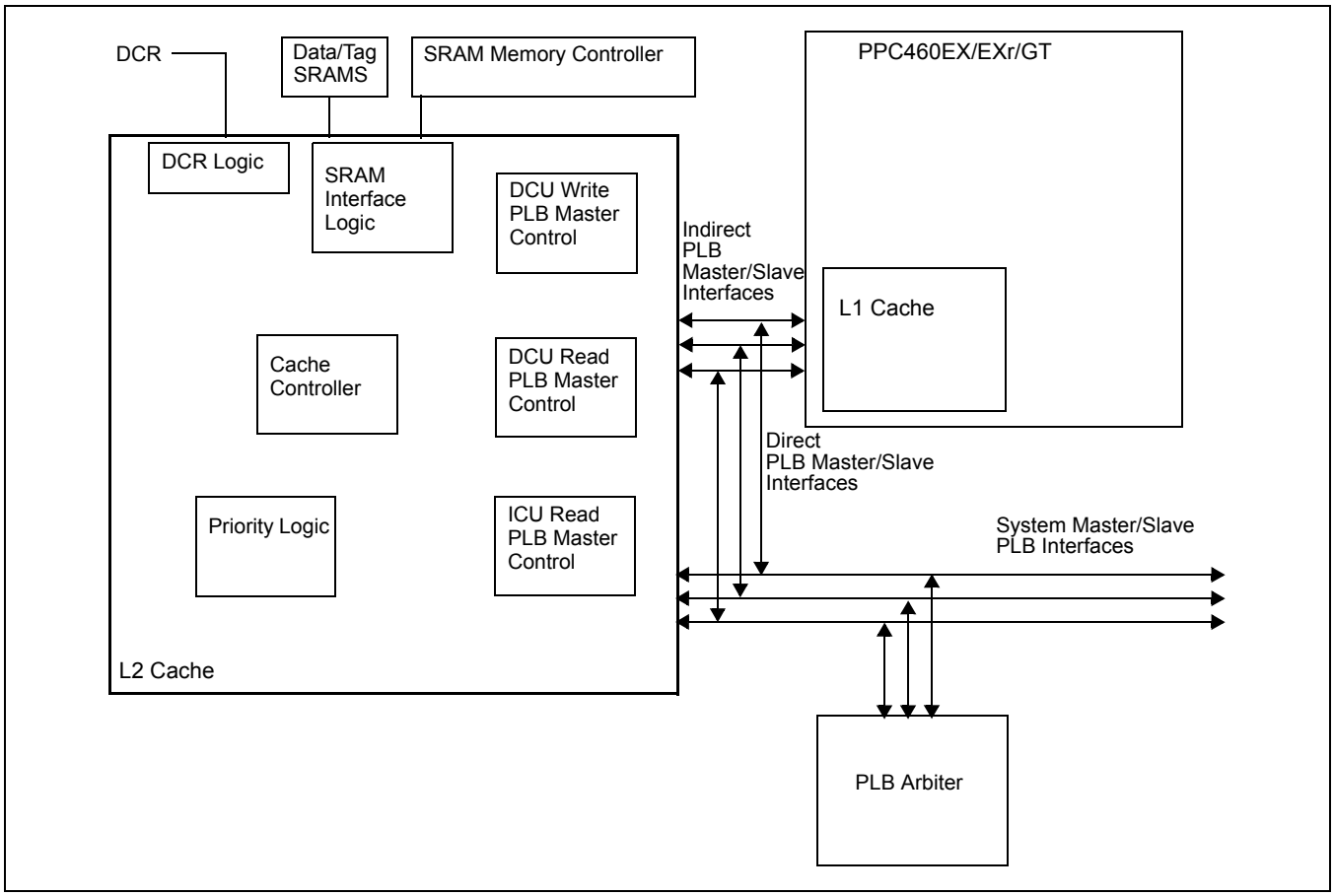
The L2C0 has an interface to allow the internal SRAM memory controller to take control of the SRAM data arrays associated with the L2C0 if L2 cache has been disabled. This enables the SRAM data arrays to be used for either the L2C0 function or SRAM memory.

When configured as L2 cache, the L2 tag is searched while a request is presented to the PLB arbiter. If there is a read hit, the L2 services the request and the PLB arbiter request is canceled. When the L2 cache is disabled, all PLB requests are passed through to the PLB arbiter.

7.1 L2 Cache Features

- 256 KB, 32-byte line, 36-bit addressing to be used with the CPU
- Increases CPU performance by decreasing data latency during L1 cache misses
- Write-through cache design with 4-way set associativity and full LRU replacement algorithm
- Decreases PLB traffic caused by L1 cache misses
- Provides unified data and instruction with separate data and instruction cache control
- Uses cacheable signals from each PLB interface to determine if an access should be cached in the L2C0
- Fast PPC440 processor read data acknowledge (RdDAck) with PLB address acknowledge (AddrAck) option on L2C0 hits (enabled/disabled through software)
- All cacheable PPC460EX/EXr/GT processor requests start with an L2C0 priority check
  - Cache hit is determined one PLB clock cycle after L2 cache priority.
  - Cache read data starts returning 2 PLB clock cycles after L2 cache priority if the fast PPC440 RdDAck option is enabled, or 4 PLB clock cycles after L2 cache priority if fast PPC440 RdDAck is disabled
- SRAM data arrays may be used by either L2 cache or by the SRAM memory controller, but not mixed
  - Enabling the L2 cache adds a one cycle PLB clock penalty for all PPC440 requests to the PLB
- Supports PLB address pipelining and is compatible with PLB arbiters supporting maximum of 4-deep read pipelining, 2-deep write pipelining
- Runs at PLB bus frequency
- Debug support
  - Each way may be independently disabled
  - Trap address of first parity error
  - Tag and cache RAM contents accessible through DCR

Figure 7-1. L2 Cache Block Diagram



7.2 L2 Cache Operations

This section discusses various connectivity issues as well as functional aspects of the chip.

7.2.1 Connection to the PPC440 Processor

L2C0 has three master-side PLB slave ports for connection to the corresponding processor's three PLB masters. The processor's PLB interface signals are either multi-dropped to both the L2C0 and the PLB arbiter, or they are redriven by the L2C0.

7.2.2 Connection to the PLB

L2C0 has three PLB master ports, replicating the portion of the PLB signals redriven through the L2C0 and one PLB slave port. The L2 has logic to handle read data returning simultaneously to both of the PLB read masters (L2C0_CFG[DOI]) but the logic has not been verified.

7.2.3 Connection to the DCR

The connection to the DCR bus allows for the configuration/debug of the L2C0.

7.2.4 Typical PLB Write Access

The DCU write can issue three types of requests: word, 4-word line, and 8-word line. Any of these requests that have the TLB cache inhibited attribute = 0 are considered L2-cacheable, processor write requests.

All L2-cacheable, PPC440 processor write requests must win L2 priority before they are presented to the L2 cache logic and the PLB arbiter in the following cycle. All other processor write requests are presented to the PLB arbiter one cycle after assertion by the processor. All writes proceed to the PLB because of the write-through policy. The L2C0 processes one L2-cacheable write hit at a time. Processor writes are cached if they hit in the L2C0. Processor writes that miss the L2C0 are not cached.

When the L2 mode is enabled (L2C0_CFG[L2M] = 1), the L2C0 logic gates the DCU PLB write request as it passes through the L2C0 and forwards it to the PLB arbiter. If there is a pending read miss for the same line (I-cache or D-cache read), the L2C0 aborts a pending write request to the PLB, and internally re-enable the write request to the PLB arbiter later. Aborting the pending cached L2C0 write improves read performance.

When L2 mode is disabled (L2C0_CFG[L2M] = 0), the L2C0 logic passes the DCU write request to the PLB arbiter without any delay.

Any errors reported by a slave device resulting from a write transaction (MWrErr, MIRQ) are passed direct to the processor. The processor responds to bus errors with a machine check exception. The L2C0 invalidates the L2 cache line in response to a bus time out. A bus time out is signaled on the PLB with the master write error signal (MWrErr).

MWrErr is asserted for errors synchronous to the transaction. The master IRQ (MIRQ) is asserted for errors that are asynchronous to the PLB transaction. MIRQ can be asserted for posted writes many PLB cycles after the transaction completed on the bus.

The L2C0 can pipeline up to two writes. However, it can only service one write hit at a time.

7.2.5 Typical PLB Read Access

On an L2 cache hit, the L2C0 provides the data. On a miss, the L2C0 reads the primary memory (typically DDR) and caches the data.

The DCU read can issue three types of requests: word, 4-word line, and 8-word line. Only 8-word line requests that have the TLB cache inhibited attribute set to 0 (I = 0) are considered L2-cacheable, DCU read requests.

The ICU read can issue 2 types of requests: 8-word line and fixed-length burst of 1-to-3 8-word lines starting on an 8-word line boundary. Any of these requests that have the TLB cache inhibited attribute = 0 are considered L2-cacheable, ICU read requests.

All L2-cacheable read requests must win L2 priority before they are presented to the L2 cache logic and the PLB arbiter in the following cycle. All other read requests are presented to the PLB arbiter one cycle after assertion by the processor. On an L2 hit, the PLB request is aborted one cycle after being asserted to the PLB arbiter. On an L2 miss, a cache way is selected based on the LRU algorithm and invalidated if necessary. The data returned on a cacheable request is passed to the processor and placed in the L2 cache. While the last part of the cache line is written into the data arrays, the tag is also written valid. The penalty for forwarding a request to the PLB on a cache miss is one cycle (or more depending on L2 activity).

When the L2 mode is enabled (L2C0_CFG[L2M] = 1), the L2C0 logic gates the DCU read requests as they pass through the L2C0 and forwards them to the PLB arbiter. There are situations where the L2C0 aborts the read request to the PLB, and might internally re-enable the request to the PLB arbiter later. These conditions are:

1. L2-cacheable I-Cache/D-cache read miss retried when there is a pending write for the same line
2. L2-cacheable I-Cache/D-cache read miss retried when there is a pending I-Cache or D-cache read for a line with the same index
3. L2-cacheable I-Cache/D-cache read hit retried when there is a pending I-Cache/D-cache read miss

When L2 mode is disabled (L2C0_CFG[L2M] = 0), the L2C0 logic passes processor read request through the L2C0 to the PLB arbiter without a delay.

The L2C0 can pipeline up to four read misses for each processor read to the PLB. The L2C0 also blocks a read hit when a read miss is pending on the same processor read.

The processor can accept returning read data faster than the PLB specification allows. To take advantage of this, the L2C0 has the capability of returning AddrAck and RdDAck together two cycles after the processor asserts PLB request. This option is enabled/disabled through DCR control (L2C0_CFG[FRAN]). If this option is disabled, RdDAck is asserted a minimum of two PLB clock cycles after AddrAck.

Cacheable read misses that result in data being returned from memory with an error indicated (MRdErr asserted) pass directly to the processor, in which case the data is not cached in L2C0. The master read error (MRdErr) is asserted synchronous to the PLB read transaction. The processor responds to a MRdErr bus error with a machine check exception.

7.2.6 Non Cacheable Requests

Non cacheable processor read and write requests do not normally search the L2 cache. These requests are passed to the PLB arbiter. However, when L2C0_CFG[SMCM] = 1 and L2C0_CFG[ICU] = 1, all write requests search the L2 cache and update the L2 copy of memory when an L2 hit occurs. This is to assist with self-modifying code.

7.2.7 L2 Cache Coherency

Coherency between the L2 cache and memory is managed via software or hardware snoop mechanism.

Note: Coherency between the L1 and memory must be managed via software. See *Section 7.4.4 Invalidating the L1 and L2 Cache* on page 201.

Software L2 cache coherency is done using the Invalidate Command, L2C0_CMD[INV], or the hardware clear command, L2C0_CMD[HCC]. The invalidate command invalidates four ways at a specified index. The hardware clear command invalidates the entire L2 cache. Because the L2 cache is write through, invalidating the cache line is sufficient to maintain coherency.

Hardware L2 cache coherency uses the PLB slave port on the low latency (LL) PLB slave segment to maintain coherency between the L2 cache and system memory for all non-processor initiated cacheable writes to system memory on LL PLB slave segment. This port is passive in that it never responds to a request. In the case of a snoop hit, the corresponding line(s) are invalidated. Since the L2 cache is write through, invalidating the line is sufficient to maintain coherency between the L2 cache and memory on the LL slave segment. Memory address ranges snooped on the LL slave segment are programmable via the L2C0_SNP0 and/or L2C0_SNP1 registers.

Note: Write accesses on the high bandwidth (HB) slave segment are not snooped by the hardware snoop mechanism.

Because the hardware snoop mechanism only snoops write accesses on the LL slave segment, all memory accesses requiring coherency must be made on the LL slave segment. To ensure coherency between the L2 cache and the DDR memory, the DDR memory must be accessed using the address range assigned to the DDR on the LL bus. Refer to the data sheet for the memory map.

7.2.8 Data Array Parity

The L2 cache data array is parity protected when L2C0_CFG[CPC] = 1. Data parity is checked on all reads that hit in the L2C0. The detection of a read data parity error results in the read data being returned to the processor with MRdErr asserted—coincident with the assertion of MRdDAck for the associated data for which the error was detected. The L2C0 traps the address of the error and invalidated the L2 cache line. The processor discards the data and does not pass this data to the GPR or L1 cache. The MRdErr causes the processor to generate a machine check. The machine check handler can check the address that the L2C0 trapped and log the error. Once the processor returns (rfi) from the machine check handler, the load that generated the failed read is attempted again. This load causes an L2 cache miss as the L2C0 invalidated the L2 cache line prior to the machine check.

7.2.9 Tag Array Parity

The L2 cache tag array is parity protected when L2C0_CFG[TPC] = 1. Tag parity is included in the tag compare. Thus, a tag parity error results in a tag miss. The L2C0 responds to this tag miss as it does to any L2 cache miss. The data is accessed from primary memory (typically DDR). Eventually, the way with a tag parity error is ages out of the L2 cache as a result of the least recently used (LRU) replacement algorithm. Tag parity soft errors are self correcting and do not generate a MRdErr or cause the processor to vector to a machine check.

7.2.10 Performance Monitoring

The L2 cache provides logic to enable performance monitoring. The I-fetch, D-fetch, and D-store interfaces can be monitored one at a time. Selection of interface monitoring is controlled by performance monitoring multiplex control (L2C0_CFG[PMUX]). See *L2 Cache Configuration Register (L2C0_CFG)* on page 192 for details.

There are three counters that monitor the number of cycles run, the number of requests, and the number of hits that occur on the selected interface. Each of these counters are 32 bits long and can be read out through the DCR interface by a diagnostic command in L2C0_CMD. The counters can be stopped, started, or reset by controls in the L2C0_CMD register. When the cycle counter reaches a value of 0xFFFFFFFF, all counters are automatically stopped. The L2C0_SR register has a bit that indicates the status of the counters (running or stopped). An interrupt condition can also be set to occur when the counters stop.

For example, to count I-fetch occurrences the following steps would be performed:

1. Set L2C0_CFG[PMUX] to select instruction accesses (ICU)
2. Set L2C0_CMD[STPC] to ensure counters are stopped
3. Set L2C0_CMD[RPMC] to reset performance counters
4. Set L2C0_CMD[STRC] to start the counters
5. Set L2C0_CFG[PMIM] to allow an interrupt
6. Wait for interrupt or poll L2C0_SR[PCS] or set L2C0_CMD[STPC]
7. Issue a DIAG performance cycle counter read to determine the number of cycles counted
8. Issue a DIAG performance request counter read to determine the number of instruction accesses (ICU)
9. Issue a DIAG performance hit counter read to determine the number of instruction accesses (ICU) that hit in the L2C0

For instruction accesses (ICU), the counter increments once for each 8-word cache line request processed, and once for each fixed-length burst processed. For data loads (DCU), the counters increment once for each 8-word cache line request processed. For data stores (DCU), the counters increment once for each 8-word cache line request, and once for each PLB cycle store request.

7.2.11 Power Management

The SRAM tag and data arrays are clock gated. There is no power management of the L2C0 logic.

7.2.12 L2C0 Disabled Operation

If the L2C0 is disabled (L2C0_CFG[L2M] = 0), there is no clock penalty for processor access requests to the PLB arbiter. If L2C0_CFG[L2M] = 0, the internal SRAM controller can be configured to use the 256KB of SRAM dedicated to the L2 use. If L2C0 is enabled (L2C0_CFG[L2M] =1), the L2, then the internal SRAM controller cannot be used. If the internal SRAM controller is enabled and configured, it can accept cycles from any PLB devices besides the processor.

7.3 L2 Cache Registers

The L2 cache controller registers are device control registers (DCRs), which are addressed by a 10-bit field in the move from device control registers (**mfdcr**) and the move to device control registers (**mtdcr**) instructions. The PPC440 processor executes these instructions.*Table 7-1* lists all L2C0 registers.

Table 7-1. L2 Cache Register Summary

Mnemonic	Register	DCR Number	Access	Page
L2C0_CFG	L2 Cache Configuration	0x0030	R/W	192
L2C0_CMD	L2 Cache Command	0x0031	R/W	195
L2C0_ADDR	L2 Cache Address	0x0032	R/W	196
L2C0_DATA	L2 Cache Data	0x0033	R	198
L2C0_SR	L2 Cache Status	0x0034	R	198
L2C0_REVID	L2 Cache Revision ID	0x0035	R	199
L2C0_SNP0	L2 Cache Snoop Register 0	0x0036	R/W	199
L2C0_SNP1	L2 Cache Snoop Register 1	0x0037	R/w	199

7.3.1 L2 Cache Configuration Register (L2C0_CFG)

Figure 7-2 on page 193 describes all L2C0_CFG register bits. However, the following information provides details about certain bits.

For L2C0_CFG[ICU] instruction cache unit enable, the following items apply:

- L2C0_CFG[L2M] must be set for L2C0_CFG[ICU] to be set.
- While L2C0_CFG[ICU] = 1, bits L2C0_CFG[DCW], L2C0_CFG[FRAN], L2C0_CFG[NAM], L2C0_CFG[SMCM] and L2C0_CFG[NBRM] must not be changed.

- While changing the state of L2C0_CFG[ICU] it is recommended that there be an **msync** instruction before and after the **mtdcr** instruction that modifies L2C0_CFG register.

For L2C0_CFG[DCU] data cache unit enable, the following items apply:

- L2C0_CFG[L2M] must be set for L2C0_CFG[DCU] to be set.
- While L2C0_CFG[DCU] = 1, bits L2C0_CFG[DCW], L2C0_CFG[FRAN], L2C0_CFG[NAM], L2C0_CFG[SMCM], and L2C0_CFG[NBRM] must not be changed.
- While changing the state of L2C0_CFG[DCU], it is recommended that there be an **msync** instruction before and after the **mtdcr** instruction that modifies L2C0_CFG register.

If L2C0_CFG[NAM] = 0, requests are presented on the PLB while the L2 is searched. Depending upon the results of the L2 search, the PLB request may be aborted. Setting L2C0_CFG[NAM] = 0 yields the highest performance. When L2C0_CFG[NAM] = 1, requests will not be presented to the PLB until the L2 search is complete. Setting L2C0_CFG[NAM] = 1 prevents the L2 from placing a request on the PLB and then aborting it.

If L2 cache is enabled for I-cache self modifying code mode (L2C0_CFG[SMCM] = 1), DCU write requests (cacheable and non-cacheable) that hit in the L2 cache update the L2 cache. This behavior is not affected by the state of L2C0_CFG[DCU].

If L2C0_CFG[NBRM] = 0 and a request has been aborted by the L2C0, any of the following conditions cause the request to not be processed by the L2C0 priority logic:

- L2-cacheable write miss retried and there is a pending read miss for the same line (I-cache or D-cache read)
- L2-cacheable I-cache/D-cache read miss retried and there is a pending write for the same line
- L2-cacheable I-cache/D-cache read miss retried and there is a pending I-cache or d-cache read for a line with the same index
- L2-cacheable I-cache/D-cache read hit retried and there is a pending I-cache/D-cache read miss

The normal operating mode is L2C0_CFG[NBRM] = 0. This saves power because the tag and data SRAMs are not clocked repeatedly by the retried request until the condition is cleared. This bit should only be enabled if a problem is found in the default disabled mode.

Figure 7-2. L2 Cache Configuration Register (L2C0_CFG)

0	L2M	L2 Mode 0 Set SRAM for use by internal SRAM controller 1 Set SRAM use as L2 cache	
1	ICU	Instruction cache unit enabled 0 Instruction cache unit does not use L2 1 Instruction cache unit enabled to use L2	
2	DCU	Data cache unit enabled 0 Data cache unit does not use L2 1 Data cache unit enabled to use L2	
3:6	DCW	Disable cache way 0000 Enable ways 0, 1, 2 and 3 0001 Enable ways 0, 1, and 2 0011 Enable ways 0 and 1 0111 Enable way 0	All other combinations reserved
7	TPC	Tag Parity Check Enable 0 Tag parity check disabled 1 Tag parity check enabled	

8	CPC	Cache Parity Check Enable 0 Cache parity check disabled 1 Cache parity check enabled	
9	DOI	Allocation of Data Over Instruction 0 Instruction allocated over data 1 Data allocated over instruction	Used when the I-side and the D-side of the L2 PLB ports can support concurrent transfer to determine the priority of the port to be allocated. If both ports are returning allocate data at the same time, only one can be placed in the L2.
10	FRAN	Fast Read Acknowledge Enable 0 Fast read acknowledge disabled 1 Fast read acknowledge enabled	For best L2 performance set L2C0_CFG[FRAN] = 1.
11:12	SS	SRAM Size 00 256KB Other combinations are reserved.	
13	CPIM	Cache Parity Error Interrupt Mask 0 Cache parity error interrupt disabled 1 Cache parity error interrupt enabled	
14	TPIM	Tag Parity Error Interrupt Mask 0 Tag parity error interrupt disabled 1 Tag parity error interrupt enabled	
15	LIM	LRU Code Point Error Interrupt Mask 0 LRU code point error interrupt disabled 1 LRU code point error interrupt enabled	
16		Reserved	
17:19	PMUX	Performance Monitoring Mask Control 000 Monitor snooping 001 Monitor I-fetching 010 Monitor D-fetching 011 Monitor D-storing 100 Reserved 101 Reserved 110 Reserved 111 Reserved	
20	PMIM	Performance Monitor Interrupt Mask 0 Performance monitor interrupt disabled 1 Performance monitor interrupt enabled	
21	TPEI	Tag Parity Error Inject 0 Tag parity error inject disabled 1 Tag parity error inject enabled	
22	CPEI	Cache Parity Error Inject 0 Cache parity error inject disabled 1 Cache parity error inject enabled	
23	NAM	No Abort Mode 0 No abort mode disabled 1 No abort mode enabled	
24	SMCM	Self Modifying Code Mode 0 Self modifying code mode disabled 1 Self modifying code mode enabled	
25	NBRM	No Block Request Mode 0 No block request mode disabled 1 No block request mode enabled	
26	SNPCI	Snoop Cache Inhibit Requests mode 0 Disabled 1 Enabled	When enabled, the L2 snoops PLB write requests within the range(s) specified by the L2C0_SNP0 and/or L2C0_SNP1 registers regardless whether the address is cacheable.

27	SNP440	Snoop 440 Write Requests mode 0 Disabled 1 Enabled	When enabled, the L2C0 snoops cache-able PPC440 PLB write requests within the range(s) specified by the L2C0_SNP0 and/or L2C0_SNP1 registers.
28	RDBW	Read Byte Write 0 Disabled 1 Enabled (required)	RDBW asserts the internal SRAM array byte write enable signals on reads.
29:31		Reserved	

7.3.2 L2 Cache Command Register (L2C0_CMD)

The following figure describes L2C0_CMD register bits. Only one command may be issued at a time. The only exception is that the L2C0_CMD[CCP] and L2C0_CMD[CTE] may be issued together. L2C0_CMD[CLR], L2C0_CMD[HCC], L2C0_CMD[DIAG], and L2C0_CMD[INV] commands must read L2C0_SR[CC] after being issued, to check for completion. If a L2C0_CMD[CLR], L2C0_CMD[HCC], L2C0_CMD[DIAG], or L2C0_CMD[INV] command is issued, the L2C0_SR[CC] bit is cleared and then set later, when the command is complete. If a new command is issued before a prior command completes, the results are unpredictable. Command bits automatically reset after the command is issued.

Note: All addresses used by the L2C0 are physical PLB bus addresses. A 36-bit PLB address is required by commands such as the Invalidate Command. Diagnostic commands such as read trapped address return the 36-bit PLB addresses. Only 36 bits of PLB address are needed to specify an address because the L2 cache and processor memory accesses are within the range 0x00000000_00000000–0x0000000F_FFFFFFFF

Figure 7-3. L2 Cache Command Register (L2C0_CMD)

0	CLR	Clear Command 0 Clear command disabled 1 Clear command enabled	L2C0_ADDR bits 16:26 are used to clear the tag index. Setting L2C0_CMD[CLR] to 1 causes all ways at the specified index to be set invalid with good parity. The LRU bits are also cleared.
1	DIAG	Diagnostic Class of Commands 0 Diagnostic class of commands disabled 1 Diagnostic class of commands enabled	For diagnostic command description see *L2 Cache Address Register (L2C0_ADDR)* on page 196.
2	INV	Invalidate Command 0 Invalidate command disabled 1 Invalidate command enabled	This command invalidates the address specified in the L2C0_ADDR register.
3	CCP	Clear Cache Parity Error 0 Clear cache parity error disabled 1 Clear cache parity error enabled	This command resets the cache trap address and way registers within the L2 such that they can trap a new error.
4	CTE	Clear Tag Error 0 Clear tag error disabled 1 Clear tag error enabled	This command resets the tag trap address and way registers within the L2 such that they can trap a new error.
5	STRC	Start Performance Monitor Counters 0 Start performance monitor counters disabled 1 Start performance monitor counters enabled	
6	STPC	Stop Performance Monitor Counters 0 Stop performance monitor counters disabled 1 Stop performance monitor counters enabled	
7	RPMC	Reset Performance Monitor Counters 0 Reset performance monitor counters disabled 1 Reset performance monitor counters enabled	

8	HCC	Hardware Clear Command 0 Hardware clear command disabled 1 Hardware clear command enabled	Setting L2C0_CFG[HCC] = 1 causes all indexes and ways of the cache to be cleared by hardware. Before issuing this command, the L2C0_ADDR field must be set to 0.
9:31		Reserved	

7.3.2.1 Example of Invalidate Command (L2C0_CMD[INV])

To invalidate 36-bit PLB address 0x1_2345_6789 out of the L2 Cache:

1. Set L2C0_ADDR = 0x23456781. L2C0_ADDR[28:31] contains the upper 4 bits of the 36-bit PLB address or upper address bits 28:31. L2C0_ADDR[0:27] contains bits 0:27 of the lower 32-bit PLB address.
2. Load L2C0_CMD with 0x20000000.
3. Poll L2C0_SR[CC] until set to 1 by L2C0.

7.3.2.2 Example of Clear Command (L2C0_CMD[CLEAR])

To clear (invalidate all four ways) at tag index 1 (address 16:26 = 0b0000_0000_001):

1. Load L2C0_ADDR with 0x00000020
2. Load L2C0_CMD with 0x80000000
3. Poll L2C0_SR[CC] until set to 1 by L2C0.

7.3.2.3 Example of Read Trapped Address Diagnostic Command (L2C0_CMD[DIAG])

1. Set L2C0_ADDR = 0x42000000.
2. Set L2C0_CMD = 0x40000000.
3. Poll L2C0_SR[CC] until set to 1 by L2C0.
4. Read L2C0_DATA. L2C0_DATA[28:31] contains the upper 4 bits of the 36-bit PLB address or upper address bits 28:31. L2C0_DATA[0:27] contains bits 0:27 of the lower 32 bit PLB address.

7.3.3 L2 Cache Address Register (L2C0_ADDR)

The following figure describes L2C0_ADDR register bits. *Table 7-2* describes the diagnostic class of commands. Bits 16:26 are used for the tag index to clear command. Bits 0:31 must be set to 0 for a hardware clear command. For an invalidate command, bits 0:27 hold bits 0:27 of the lower PLB address and bits 28:31 hold bits 28:31 of the upper PLB address.

Figure 7-4. L2 Cache Address Register (L2C0_ADDR)

0:31	ADDR	Address	

Table 7-2. Diagnostic Class of Commands use of L2C0_ADDR Register

DIAG Command	L2C0_ADDR Bits
Tag Read Way	0:7: 1000 1000 8:11: Reserved (set to 0) 12:15: 1000 Way 0 0100 Way 1 0010 Way 2 0001 Way 3 Other: Reserved 16:26: Tag Index 27:31: Reserved (set to 0) Valid, upper addr 28:31, address 0:17, parity is placed in L2C0_DATA[0:23]
Tag Read LRU	0:7: 1000 0100 8:15: Reserved (set to 0) 16:26: Tag Index 27:31: Reserved (set to 0) LRU data (0:5) is placed in L2C0_DATA[0:5]
Tag Read Trapped Address	0:7: 1000 0010 8:31: Reserved (set to 0) Trapped address (16:26) is placed in L2C0_DATA[0:10]
Tag Read Trapped Way	0:7: 1000 0001 8:31: Reserved (set to 0) Trapped way (0:3) is placed in L2C0_DATA[0:3]
Cache Read Word	0:7: 0100 1000 8:11: Reserved (set to 0) 12:15: 1000 Way 0 0100 Way 1 0010 Way 2 0001 Way 3 Other: Reserved 16:26: Cache Index 27:29: Word specification 30:31: Reserved (set to 0) Read data (0:31) is placed in L2C0_DATA[0:31]
Cache Read Parity	0:7: 0100 0100 8:11: Reserved (set to 0) 12:15: 1000 Way 0 0100 Way 1 0010 Way 2 0001 Way 3 Other: Reserved 16:26: Cache Index 27: quad word specification 28:31: Reserved (set to 0) Parity data (0:15) is placed in L2C0_DATA[0:31]
Cache Read Trapped Address	0:7: 0100 0010 8:31: Reserved (set to 0) Upper address (28:31), address (0:27) is placed in L2C0_DATA[0:31]

Table 7-2. Diagnostic Class of Commands use of L2C0_ADDR Register (Continued)

DIAG Command	L2C0_ADDR Bits
Cache Read Trapped Way	0:7: 0100 0001 8:31: Reserved (set to 0) Trapped way (0:3) is placed in L2C0_DATA[0:3]
Performance Cycle Counter Read	0:7: 0010 1000 8:31: Reserved (set to 0) Performance cycle counter is placed in L2C0_DATA[0:31]
Performance Request Counter Read	0:7: 0010 0100 8:31: Reserved (set to 0) Performance request counter is placed in L2C0_DATA[0:31]
Performance Hit Counter Read	0:7: 0010 0010 8:31: Reserved (set to 0) Performance hit counter is placed in L2C0_DATA[0:31]

7.3.4 L2 Cache Data Register (L2C0_DATA)

Figure 7-5. L2 Cache Data Register (L2C0_DATA)

0:31		Data	

7.3.5 L2 Cache Status Register (L2C0_SR)

Figure 7-6. L2 Cache Status Register (L2C0_SR)

0	CC	Command Complete 0 Command issued in L2_CMD not complete 1 Command issued in L2_CMD complete	When L2C0_CMD[CLR], L2C0_CMD[HCC], L2C0_CMD[DIAG], or L2C0_CMD[INV] is issued, L2C0_SR[CC] is cleared and then set when the command is complete.
1	CPE	Cache Parity Error 0 No cache parity error present 1 Cache parity error present	If L2C0_SR[CPE] = 1 and error is unmasked, an interrupt will be presented
2	TPE	Tag Parity Error 0 No tag parity error present 1 Tag parity error present	If L2C0_SR[TPE] = 1 and error is unmasked, an interrupt will be presented
3	LRU	LRU Code Point Error 0 No LRU error present 1 LRU error present	If L2C0_SR[LRU] = 1 and error is unmasked, an interrupt will be presented
4	PCS	Performance Counter Stopped 0 Performance counter is running 1 Performance counter is stopped	If L2C0_SR[PCS] = 1 and condition is unmasked, an interrupt will be presented
5:31		Reserved	

7.3.6 L2 Cache Revision ID Register (L2C0_REVID)

Figure 7-7. L2 Cache Revision ID Register (L2C0_REVID)			
0:31	REVID	Core Revision ID	

7.3.7 L2 Cache Snoop Register0:1 (L2C0_SNP0:L2C0_SNP1)

Figure 7-8. L2 Cache Snoop Register 0:1 (L2C0_SNP0:L2C0_SNP1)			
0:3	UA	Upper Base Address 0x0 - 0x7 map to the low latency PLB slave segment. The DDR memory on the low latency PLB slave segment have an upper 4-bit address within the range 0x0 - 0x3.	L2C0_SNP0:L2C0_SNP1[0:15] specifies the starting address of the snooping region. L2C0_SNP0:1[0:3] is the upper 4 bits of the 36-bit physical PLB address or bit positions2^{35} - 2^{32}. L2C0_SNP0:1[4:15] are bits 0:11 of the lower 32-bit PLB address or bit positions 2^{31} - 2^{20}. Example: 36-bit address: 0x123400000. LC0_SNP0:1[0:3]=0x1 LC0_SNP0:1[4:15]=0x234
4:15	BA	Base Address	
16:19	SSR	Size of Snooping Region 0000 1MB 0001 2MB 0010 4MB 0011 8MB 0100 16MB 0101 32MB 0110 64MB 0111 128MB 1000 256MB 1001 512MB 1010 1GB 1011 2GB 1100 4GB 1101 8GB 1110 16GB 1111 32GB	
20	ESR	Enable Snoop Region 0 Snoop region disabled 1 Snoop region enable	See erratum on the snoop mechanism before using.
21:31		Reserved	

Note: The base address must be aligned on a boundary that matches the size of the snooping region.

7.4 Reset and Initialization

The following sections provide information about enabling and disabling the L2 cache for the PPC440 processor. A subsequent note discusses SRAM memory controller-to-PLB mode.

7.4.1 Enabling the L2 Cache

Before the L2 cache can be enabled, the processor must reset the tag array using the hardware clear command (L2C0_CMD[HCC]) or clear commands (L2C0_CMD[CLR]). The following code provides an example of both methods. The L2C0_CMD[HCC] method is preferred because it executes in the shortest amount of time.

Preferred method, using the L2C0_CMD[HCC] command:

1. Disable the SRAM0 controller. Set SRAM0_SB0CR = 0, SRAM0_SB1CR = 0, SRAM0_SB2CR = 0 and SRAM0_SB3CR = 0.
2. Invalidate any MMU TLB entries used assigned to SRAM0 memory.
3. Set L2C0_CFG[RDBW] = 1.
4. Enable L2_MODE, L2C0_CFG[L2M] = 1.
5. Set L2C0_ADDR = 0x00000000.
6. Issue the HCLEAR command, L2C0_CMD[HCC] = 1.
7. Poll L2C0_SR[CC] until set.
8. Clear cache parity errors, L2C0_CMD[CCP] = 1.
9. Poll L2C0_SR[CC] until set.
10. Clear tag parity errors, L2C0_CMD[CTE] = 1.
11. Poll L2C0_SR[CC] until set.
12. Execute an **msync** instruction.
13. Enable processor instruction and data L2 Cache access as desired, L2C0_CFG[ICU] = 1 and L2C0_CFG[DCU] = 1.
14. Execute an **msync** instruction.

Alternate method, using the L2C0_CMD[CLR] command:

1. Disable SRAM0 controller. Set SRAM0_SB0CR = 0, SRAM0_SB1CR = 0, SRAM0_SB2CR = 0 and SRAM0_SB3CR = 0.
2. Invalidate any MMU TLB entries used assigned to SRAM0 memory.
3. Set L2C0_CFG[RDBW] = 1.
4. Enable L2_MODE, L2C0_CFG[L2M] = 1.
5. Clear each tag index using the CLEAR command, L2C0_CMD[CLR].

```
TagIndex = 0x00000000//Set TagIndex variable to zero.

for (i=0; i <2048; i++) {// do once for each tag index
    set L2C0_ADDR = TagIndex // The tag index is L2C0_ADDR[16:26] for the clear command.
    set L2C0_CMD[CLR]= 1 //issue CLEAR command
    poll L2C0_SR[CC] until set by L2C0
    TagIndex = TagIndex + 0x00000020 // Increment tag index by 1.
}
```

6. Clear cache parity errors, L2C0_CMD[CCP] = 1.
7. Poll L2C0_SR[CC] until set.
8. Clear tag parity errors, L2C0_CMD[CTE] = 1.
9. Poll L2C0_SR[CC] until set.
10. Execute an **msync** instruction.
11. Enable processor instruction and data L2 Cache access as desired, L2C0_CFG[ICU] = 1 and L2C0_CFG[DCU] = 1.
12. Execute an **msync** instruction

7.4.2 Disabling the L2 Cache

To stop the processor from using the L2 cache, reset L2C0_CFG[ICU] and L2C0_CFG[DCU]. It is recommended that the **msync** instruction be issued before and after the **mtdcr** for L2C0_CFG.

```
msync
write L2C0_CFG[L2M] = 0
msync
```

To re-enable the L2 cache, follow the steps described at *Enabling the L2 Cache* on page 200.

7.4.3 SRAM Memory Controller to PLB4 Mode

To access the L2Cache as an SRAM, configure the SRAM0 registers. See *SRAM Controller* on page 203.

7.4.4 Invalidating the L1 and L2 Cache

Invalidating cache lines in the L1 cache using L1 data cache instructions (**dcbi**, **dcbf**, etc.) does not invalidate the data from the L2 cache. It is important to invalidate the addresses in the L1 cache first and then invalidate the same addresses using the L2 cache invalidate command. Failure to invalidate both caches can cause coherency problems.

8. SRAM Controller

The internal SRAM Controllers (SRAM0 and SRAM1) provide access to the L2Cache SRAM and on-chip SRAM (OCM), and the Processor Local Bus (PLB). The 256KB L2Cache SRAM is accessed by means of SRAM0 when the L2Cache Controller is disable (L2C0_CFG[L2M] = 0). The 64KB OCM is accessed by means of SRAM1.

The ISC supports the following features:

- L2Cache SRAM, four banks (Bank0:3) of 64KB (128 bits wide)
- OCM, one bank (Bank 0) of 64KB (128 bits wide)
- 128-bit slave attachment addressable by any PLB master
- Transfers by PLB slave cycles:
  - Single-beat read and write (1 to 8 bytes for 64-bit masters, 1 to 16 bytes for 128-bit masters)
  - 4-word line read and write
  - 8-word line read and write
  - Double word read and write bursts for 64-bit masters
  - Quadword read and write bursts for 128-bit masters
  - Slave-terminated double word and quadword fixed length bursts
  - Master-terminated variable length bursts
- Guarded memory access on 4KB boundaries
- Data parity checking
- Data transfers occur at PLB bus speeds.
- Power management

8.1 SRAM Controller Registers

The Internal SRAM Controller registers listed in *Table 8-1* are accessed by using move to device control register (**mtdcr**) and move from device control register (**mfdcr**) instructions.

Table 8-1. SRAM Registers

Mnemonic	Register	DCR Address	Access	Page
SRAM0_SB0CR	SRAM0 Bank 0 Configuration Register	0x020	R/W	204
SRAM0_SB1CR	SRAM0 Bank 1 Configuration Register	0x021	R/W	204
SRAM0_SB2CR	SRAM0 Bank 2 Configuration Register	0x022	R/W	204
SRAM0_SB3CR	SRAM0 Bank 3 Configuration Register	0x023	R/W	204
SRAM0_BEAR	SRAM0 Bus Error Address Register	0x024	R/W	205
SRAM0_BESR0	SRAM0 Bus Error Status Register 0	0x025	R/W	205
SRAM0_BESR1	SRAM0 Bus Error Status Register 1	0x026	R/W	207
SRAM0_PMEG	SRAM0 Power Management Register	0x027	R/W	209
SRAM0_CID	SRAM0 Core ID Register	0x028	Read	209
SRAM0_REVID	SRAM0 Revision ID Register	0x029	Read	210

Table 8-1. SRAM Registers (Continued)

SRAM0_DPC	SRAM0 Data Parity Check Register	0x02A	R/W	210
SRAM1_SB0CR	SRAM1 Bank 0 Configuration Register	0x0B0	R/W	
SRAM1_BEAR	SRAM1 Bus Error Address Register	0x0B4	R/W	205
SRAM1_BESR0	SRAM1 Bus Error Status Register 0	0x0B5	R/W	206
SRAM1_BESR1	SRAM1 Bus Error Status Register 1	0x0B6	R/W	207
SRAM1_PMEG	SRAM1 Power Management Register	0x0B7	R/W	209
SRAM1_CID	SRAM1 Core ID Register	0x0B8	Read	209
SRAM1_REVID	SRAM1 Revision ID Register	0x0B9	Read	210
SRAM1_DPC	SRAM1 Data Parity Check Register	0x0BA	R/W	210

8.1.1 SRAM Bank Configuration Register (SRAMx_SBnCR)

To ensure proper address decode and translation by the SRAM controller, the four least-significant bits (lsb) of the upper 32-bit PLB address and the twenty most-significant bits (msb) of the lower 32-bit PLB address must be written to SRAMx_SBnCR[28:31] and SRAMx_SBnCR[0:19], respectively. See *Errors* on page 210 for additional programming notes.

Note 1: n in SRAMx_SBnCR is the bank number. SRAM0 for the L2Cache SRAM has four banks of 64KB (SRAM0_SB0CR–SRAM0_SB3CR). SRAM1 for the OCM has one bank of 64KB (SRAM1_SB0CR).

Note 2: When configuring the L2 cache for use as SRAM, set L2C0_CFG[ICU] = 0 and L2C0_CFG[DCU] = 0 before clearing L2C0_CFG[L2M]. Execute an **msync** instruction before and after each **mtdcr** instruction that modifies the L2C0_CFG.

SRAM0 (L2Cache SRAM) Recommended Configuration:

Execute **msync**.

Set L2C0_CFG[ICU] = 0.

Set L2C0_CFG[DCU] = 0.

Set L2C0_CFG[L2M] = 0.

Set L2C0_CFG[RDBW] = 1.

Set SRAM0_SB0CR = 0x00000984 ;base address as 0x4_0000_0000, size 64KB, read/write access.

Set SRAM0_SB1CR = 0x00010984 ;base address as 0x4_0001_0000, size 64KB, read/write access.

Set SRAM0_SB2CR = 0x00020984 ;base address as 0x4_0002_0000, size 64KB, read/write access.

Set SRAM0_SB3CR = 0x00030984 ;base address as 0x4_0003_0000, size 64KB, read/write access.

Execute **msync**.

When not using SRAM0 (L2C0_CFG[L2M] = 1), set SRAM0_SBnCR = 0x00000000.

SRAM1 (OCM) Recommended Configuration:

Set SRAM1_SB0CR = 0x00040984 ;base address as 0x4_0004_0000, size 64KB, read/write access.

Execute **msync**.

Note: The memory assigned to SRAM0 and SRAM1 should not be cached. Access to the on-chip SRAM is much faster then access to DDR or other memory, therefore, caching this memory does not significantly improve performance.

Figure 8-1. Memory Configuration (SRAMx_SBnCR)

0: 19	BAS	Base Address Select	Sets the base address for an SRAM range. The BAS field is compared to bits 0:19 of the effective address. If the effective address is within the range of the starting address plus bank size, the associated bank is enabled for the transaction. Reset value = 0x00040.
20:22	BS	Bank Size 000 Reserved 001 Reserved 010 Reserved 011 Reserved 100 64KB 101 Reserved 110 Reserved 110 Reserved	Specifies the size of the logical bank address range. Sets the number of bytes which the bank may access, beginning with the base address set in the BAS field. Reset value = 0b100.
23:24	BU	Bank Usage 00 Disabled 01 Bank is valid for read only (RO) 10 Reserved 11 Bank is valid for read/write (R/W)	Protects banks of physical devices from read or write accesses. Reset value = 0b11.
25:27		Reserved	
28:31	UA	Upper four bits of the 36-bit PLB address.	Reset value = 0x4.

When an attempt is made to write access to an address within the range of the BAS field and the bank is designated as read-only, an SRAM controller protection error occurs.

8.1.2 SRAM Bus Error Address Register (SRAMx_BEAR)

The SRAM Bus Error Address Register (SRAMx_BEAR) is a 32-bit register which contains the address of the access where a data bus error has occurred. SRAMx_BEAR is written when a data access error occurs and its contents may be locked until the SRAMx_BEAR lock bit in the SRAM Bus Error Status Register (SRAMx_BESR0 or SRAMx_BESR1) is cleared, depending on the state of the LOCK ERROR signal when the transaction was accepted. The contents of SRAMx_BEAR can be accessed using the move from device control register (**mfdcr**) and move to device control register (**mtdcr**) instructions.

Note: The upper four bits of the 36-bit PLB address is not recorded.

Figure 8-2. SRAM Bus Error Address Register (SRAMx_BEAR)

0:31	ABE	Address of Bus Error (asynchronous)	

8.1.3 SRAM Bus Error Status Register 0 (SRAMx_BESR0)

The SRAM Bus Error Status Register 0 (SRAMx_BESR0) records the occurrence and type of errors for transactions attempted on behalf of each PLB master. The contents of the SRAMx_BESR0 can be accessed using the move from device control register (**mfdcr**) and move to device control register (**mtdcr**) instructions.

The field lock bit protects fields ETn, RWSn, FLn, and ALn for master n (where n = 0, 1, 2, and 3). Upon error detection, any of master n fields may be overwritten if the field lock bit for that master is zero. If the field lock bit for a particular master has been set to zero by the DCR master, the BEAR lock bit for that master is overwritten when the next PLB error is detected, regardless of the status of the BEAR lock prior to the error. If the BEAR lock is overwritten with 0, the BEAR is overwritten on the next error detected from that master, but the BEAR cannot be overwritten if the error is due to an error caused by a different master.

Figure 8-3. SRAM Bus Error Status Register 0 (SRAMx_BESR0)

0:2	ET0	Error type for master 0 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 0 is the processor ICU.
3	RWS0	Read/write status for master 0 0 Error operation was a write operation 1 Error operation was a read operation	
4	FL0	Field lock for master 0 0 Fields are unlocked 1 Fields are locked	
5	AL0	BEAR address lock for master 0 0 BEAR address unlocked 1 BEAR address locked	
6:8	ET1	Error type for master 1 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 1 is the processor DCU read interface.
9	RWS1	Read/write status for master 1 0 Error operation was a write operation 1 Error operation was a read operation	
10	FL1	Field lock for master 1 0 Fields are unlocked 1 Fields are locked	
11	AL1	BEAR address lock for master 1 0 BEAR address unlocked 1 BEAR address locked	

12:14	ET2	Error type for master 2 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 2 is the PCI controller (PCI0).
15	RWS2	Read/write status for master 2 0 Error operation was a write operation 1 Error operation was a read operation	
16	FL2	Field lock for master 2 0 Fields are unlocked 1 Fields are locked	
17	AL2	BEAR address lock for master 2 0 BEAR address unlocked 1 BEAR address locked	
18:20	ET3	Error type for master 3 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 3 is the PCI Express 1 controller (PCIE1).
21	RWS3	Read/write status for master 3 0 Error operation was a write operation 1 Error operation was a read operation	
22	FL3	Field lock for master 3 0 Fields are unlocked 1 Fields are locked	
23	AL3	BEAR address lock for master 3 0 BEAR address unlocked 1 BEAR address locked	
24:31		Reserved	

8.1.4 SRAM Bus Error Status Register 1 (SRAMx_BESR1)

The SRAM Bus Error Status Register 1 (SRAMx_BESR1) records the occurrence and type of errors for transactions attempted on behalf of each PLB master. The contents of the SRAMx_BESR1 can be accessed using the move from device control register (**mfdcr**) and move to device control register (**mtdcr**) instructions.

The field lock bit protects fields ETn, RWSn, FLn, and ALn for master n (where n = 4, 5, 6, and 7). Upon error detection, any of master n fields may be overwritten if the field lock bit for that master is zero. If the field lock bit for a particular master has been set to zero by the DCR master, the BEAR lock bit for that master will be overwritten when the next PLB error is detected, regardless of the status of the BEAR lock prior to the error. If the BEAR lock is overwritten as a zero, the BEAR will be overwritten on the next error detected from that master, but will not be overwritten if the BEAR lock is due to an error caused by another master.

Figure 8-4. Bus Error Status Register 1 (SRAMx_BESR1)			
0:2	ET4	Error type for master 4 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 4 is the processor DCU write interface.
3	RWS4	Read/write status for master 4 0 Error operation was a write operation 1 Error operation was a read operation	
4	FL4	Field lock for master 4 0 Fields are unlocked 1 Fields are locked	
5	AL4	BEAR address lock for master4 0 BEAR address unlocked 1 BEAR address locked	
6:8	ET5	Error type for master 5 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 5 is the PCI Express 0 controller (PCIE0).
9	RWS5	Read/write status for master 5 0 Error operation was a write operation 1 Error operation was a read operation	
10	FL5	Field lock for master 5 0 Fields are unlocked 1 Fields are locked	
11	AL5	BEAR address lock for master 5 0 BEAR address unlocked 1 BEAR address locked	
12:14	ET6	Error type for master 6 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 6 is the DMA-to-PLB4 controller (DMA2P40).
15	RWS6	Read/write status for master 6 0 Error operation was a write operation 1 Error operation was a read operation	
16	FL6	Field lock for master 6 0 Fields are unlocked 1 Fields are locked	
17	AL6	BEAR address lock for master 6 0 BEAR address unlocked 1 BEAR address locked	

18:20	ET7	Error type for master 7 000 No error 001 Reserved 010 Parity error 011 Reserved 100 Protection error 101 Reserved 110 Reserved 111 Reserved	Master 7 is the I2O/DMA controller (HSDMA).
21	RWS7	Read/write status for master 7 0 Error operation was a write operation 1 Error operation was a read operation	
22	FL7	Field lock for master 7 0 Fields are unlocked 1 Fields are locked	
23	AL7	BEAR address lock for master 7 0 BEAR address unlocked 1 BEAR address locked	
24:31		Reserved	

8.1.5 SRAM Power Management Register (SRAMx_PMEG)

The SRAM Power Management Register (SRAMx_PMEG) enables the sleep function for the SRAM controller.

The internal SRAM controller is a class 2 clock and power manage device. By setting PMEN to 1, the internal SRAM controller, as a class 2 device, is able to request the clock and power management (CPM) to place it to sleep. In order for the CPM to place the internal SRAM controllers to sleep, the CPM0_ER[SRAM] for SRAM0 and CPM0_ER[OCM] bit fields for the OCM must be set to 1.

Figure 8-5. SRAM Power Management Register (SRAMx_PMEG)

0	PMEN	Power Management enable: 0 Sleep mode disabled 1 Sleep mode enabled	
1:6	PMCNT	Power Management Counter	The value (n) programmed into these register bits is a multiple of 16 clock cycles (n x 16). If PMEN is set to 1, the SRAM controller goes into sleep mode after being idle for n x 16 clock cycles.
7:10	PMDFLT	Power Management Default Wait Interval Hard wired to 1111.	This field is read-only. The hard wired value is 16 clock cycles. If PMEN is set to 1, SRAM controller goes into sleep mode after being idle for 16 clock cycles.
11:31		Reserved	

8.1.6 SRAM Core ID Register (SRAMx_CID)

The register identifies the core number for the SRAM controller. The reset value = 0.

Figure 8-6. SRAM Core ID Register (SRAMx_CID)

0:31	CID	Internal Core Device ID	

8.1.7 SRAM Revision ID Register (SRAMx_REVID)

The read-only SRAM Revision ID Register (SRAMx_REVID) identifies the revision and branch numbers, for the SRAM controller. For example, a value of 0xXXX0_0150 corresponds to a revision number of 1.50.

Figure 8-7. SRAM Revision ID Register (SRAMx_REVID)

0:11		Reserved	
12:23	REV	Revision version	Reset value = 0x001
24:31	BRANCH	Branch revision number	Reset value = 0x60

8.1.8 SRAM Data Parity Checking Register (SRAMx_DPC)

The SRAM Data Parity Checking register (SRAMx_DPC) enables data parity generation and checking if the data parity function is implemented. It controls data parity generation during write transactions and data parity checking during read transactions.

Note: After a reset, the default DPC value is 0. Before enabling parity checking (DPC=1), write the entire SRAM with data. Writing the SRAM initializes the SRAM parity.

Figure 8-8. SRAM Data Parity Checking Register (SRAMx_DPC)

0	DPC	Data Parity Check: 0 Disabled 1 Enabled	When set to 1, this bit enables data parity generation during write transactions and read transactions. When set to 0, this bit disables both functions.
1:31		Reserved	

8.2 Errors

The SRAM controller monitors two types of errors when executing PLB transfers: Protection and Data Parity errors.

8.2.1 Protection Error

A Protection error occurs when the requested read or write operation violates the bank usage programming, for example, when a write is attempted on read-only bank.

When the SRAM detects an error, it reports this error condition to the master who originated the read access. The internal SRAM controller records the type of error in the appropriate SRAMx_BESR0 or SRAMx_BESR1 bit field, and the address at which this error occurred in the SRAMx_BEAR. There is one error field for PLB masters 0–7, but only one SRAMx_BEAR.

Note: Errors for masters 8, 9, 10, and 11 are not recorded by the internal SRAM controller.

8.2.1.1 BESR Field

The address recorded by the SRAMx_BEAR depends on the Bus Error Status Register BEAR lock bit field. When the BEAR lock bit field is enabled for a master, SRAMx_BEAR contains the address due to the master's first error. When the BEAR lock bit field is disabled for a master, SRAMx_BEAR contains the address due to the master's most recent error.

8.2.2 Data Parity Error

A Data Parity error occurs when the parity of data during a read transaction does not match the parity that is generated and stored for the specific data. Generation of stored parity occurs during write transactions. The operation of parity error is identical to protect error. The SRAMx_BEAR contains the address at which parity error occurred. Unlike with the protect error, the address captured in the SRAMx_BEAR for parity error could be one address beyond where the parity error occurred. Parity errors for masters 0–7 are recorded in SRAMx_BESRn.

9. Bootstrap Operations

In preparation for booting, the PPC460EX/EXr/GT initializes register bit fields that affect system clocking, booting and other system features during a system reset. The initialization settings used in this process are determined by one of eight hard wired combinations on the three external bootstrap pins. See *Table 9-2*.

Two of the hard wired strap combinations cause the initialization setting to be read by the IIC bootstrap controller from a serial ROM attached to the IIC interface.

The other six hard wired bootstrap combinations cause the PPC460EX/EXr/GT to be initialized with predefined settings.

9.1 Bootstrap Configuration

Bootstrap values include clock ratios, PLL parameters, boot settings and several other configurations affecting NDFC, Ethernet and pin sharing. Options selected during reset are used to initialize registers throughout the PPC460EX/EXr/GT and are stored in the Serial Device Strap registers (SDR0_SDSTP0-SDR0_SDSTP3). *Table 9-1* lists the Serial Device Strap registers and the corresponding register bit fields initialized during reset.

Table 9-1. Bootstrap Register Bit Fields (Sheet 1 of 3)

Serial Device Strap Register	Serial Device Strap Register Bit Field	Bit Field Initialized During Reset	Description
SDR0_SDSTP0	SDR0_SDSTP0[ENG]	CPR0_PLLC[ENG]	Engage
	SDR0_SDSTP0[SEL]	CPR0_PLLC[SEL]	Feedback Selection
	SDR0_SDSTP0[TUNE]	CPR0_PLLC[TUNE]	PLL Tune Bits
	SDR0_SDSTP0[FBDV]	CPR0_PLLD[FBDV]	PLL Feedback Divisor
	SDR0_SDSTP0[FWDVA]	CPR0_PLLD[FWDVA]	PLL Forward Divisor A
	SDR0_SDSTP0[FWDVB]	CPR0_PLLD[FWDVB]	PLL Forward Divisor B

Table 9-1. Bootstrap Register Bit Fields (Sheet 2 of 3)

Serial Device Strap Register	Serial Device Strap Register Bit Field	Bit Field Initialized During Reset	Description
SDR0_SDSTP1	SDR0_SDSTP1[PLBEDV0]	CPR0_PLBED[PLBEDV0]	PLB Early Clock Divisor
	SDR0_SDSTP1[PLB2DV0]	CPR0_PLB2D[PLB2DV0]	PLB2 Clock Divisor
	SDR0_SDSTP1[OPBDV0]	CPR0_OPBD[OPBDV0]	OPB Clock Divisor
	SDR0_SDSTP1[PERDV0]	CPR0_PERD[PERDV0]	Peripheral Clock Divisor 0
	SDR0_SDSTP1[AHBD0]	CPR0_AHBD[AHBDV0]	AHB Clock Divisor 0
	SDR0_SDSTP1[AHBSTP]	SDR0_AHB_CFG[AHBSTP]	AHB Clock Stop
	SDR0_SDSTP1[RL]	SDR0_CP440[RL]	Boot ROM Location
	SDR0_SDSTP1[PAE]	SDR0_PCI0[PAE]	PCI Arbiter Enable
	SDR0_SDSTP1[PHCE]	SDR0_PCI0[PHCE]	PCI Host Configuration Enable
	SDR0_SDSTP1[PISE]	SDR0_PCI0[PISE]	PCI Initial Sequence Enable
	SDR0_SDSTP1[PCWE]	SDR0_PCI0[PCWE]	PCI Local CPU Wait Enable
	SDR0_SDSTP1[PPIM]	SDR0_PCI0[PPIM]	PCI Inbound Map (PIM) Settings
	SDR0_SDSTP1[SATA]	SDR0_PE0_PHY_CTL_RST [31]	PE0/SATA Selection
	SDR0_SDSTP1[SRIO]	SDR0_PE1_PHY_CTL_RST [30]	PE1/SRIO Selection
	SDR0_SDSTP1[RW]	SDR0_EBC0[RW] EBC0_B0CR[BW]	Boot ROM Width
	SDR0_SDSTP1[RS]	SDR0_EBC0[RS]	Boot ROM Size
	SDR0_SDSTP1[DBG]	SDR0_PFC0[DBG]	Debug Mode
	SDR0_SDSTP1[ZMIIM]	SDR0_ETH_CFG[ZMIIM]	ZMII Bridge Mode Selector
	SDR0_SDSTP1[GMC1BS]	SDR0_ETH_CFG[GMC1BS]	GMC Port 1 Bridge Selector
	SDR0_SDSTP1[GMC0BS]	SDR0_ETH_CFG[GMC0BS]	GMC Port 0 Bridge Selector
SDR0_SDSTP2	SDR0_SDSTP2[MEN]	SDR0_CUST0[MEN]	Multiplex GPIO or NDFC
	SDR0_SDSTP2[NE]	SDR0_CUST0[NE] NDFC0_B0CR[EN]	NDFC Enable
	SDR0_SDSTP2[NBW]	SDR0_CUST0[NBW] NDFC0_B0CR[SZ]	NDFC Boot Width
	SDR0_SDSTP2[NBP]	SDR0_CUST0[NBP] NDFC0_CR[RPG]	NDFC Boot Page
	SDR0_SDSTP2[NBAC]	SDR0_CUST0[NBAC] NDFC0_CR[ARAC]	NDFC Boot Address Cycle
	SDR0_SDSTP2[NARE]	SDR0_CUST0[NARE] NDFC0_CR[ARE]	NDFC Auto read enable
	SDR0_SDSTP2[NRB]	SDR0_CUST0[NRB] NDFC0_CR[REN]	NDFC Ready/Busy bit
	SDR0_SDSTP2[NDRSC]	SDR0_CUST0[NDRSC]	NDFC Device Reset Count
	SDR0_SDSTP2[NCG]	SDR0_CUST0[NCG]	NDFC CS Gating

Table 9-1. Bootstrap Register Bit Fields (Sheet 3 of 3)

Serial Device Strap Register	Serial Device Strap Register Bit Field	Bit Field Initialized During Reset	Description
SDR0_SDSTP3	SDR0_SDSTP3[NDRDC]	SDR0_CUST1[NDRDC]	NDFC Device Read Count

9.2 Bootstrap Options

The PPC460EX/EXr/GT has eight hard wired bootstrap options described in *Table 9-2*. The bootstrap pins are wired for the system reset default option. The six options A, B, C, D, E, and F provide predefined bootstrap configurations. Options G and H enable the IIC bootstrap controller and select the IIC address 0xA8 or 0xA4 respectively.

The six hard wired bootstrap options A, B, C, D, E, and F include configurations for booting from either an EBC or NDFC attached ROM. Options A, B, and E support booting from an EBC attached ROM with 8- or 16-bit data widths. Options C and D support booting over PCI. Option F supports booting over NDFC. *Table 9-3* contains descriptions of each bootstrap option.

If options G or H are wired, the controller reads 16 bytes from a serial ROM accessible from the IIC0 interface and stores them in the Serial Device Strap Registers. If the IIC bootstrap controller is unable to successfully read data from the serial ROM, the PPC460EX/EXr/GT defaults to Bootstrap Option A.

If the IIC bootstrap controller is disabled (strapping pins are wired for one of the options A—F), the PPC460EX/EXr/GT is not limited to the selected bootstrap configuration. After system reset, software can change the initilization settings by following the procedure described below.

9.2.1 Software Boot Configuration Procedure

1. Check the following registers for the desired configuration:
   - CPR0_PLLC
   - CPR0_PLLD
   - CPR0_PLBED
   - CPR0_PLB2D
   - CPR0_OPBD
   - CPR0_PERD
   - CPR0_AHBD
   - SDR0_AHB_CFG[AHBSTP]
   - SDR0_CP440[RL]
   - SDR0_PCI0[PAE]
   - SDR0_PCI0[PHCE]
   - SDR0_PCI0[PISE]
   - SDR0_PCI0[PCWE]
   - SDR0_PCI0[PPIM]
   - SDR0_EBC0[RW]
   - SDR0_EBC0[RS]
   - SDR0_PFC0[DBG]
   - To enable SATA, set SDR0_PE0_PHY_CTL_RST [31] =1 (PPC460EX/EXr only)
   - To enable SRIO, set SDR0_PE1_PHY_CTL_RST [31] =1 (PPC460GT only)
   - SDR0_ETH_CFG[ZMIIM]
   - SDR0_ETH_CFG[GMC1BS]
   - SDR0_ETH_CFG[GMC0BS]

If these registers are correctly configured, no further action is required and steps 2 and 3 need not be done. Otherwise, set the desired configuration and proceed with steps 2 and 3.

2. Set CPR0_ICFG[RLI] = 1. The Reload Inhibit bit preserves the configuration set in step 1 through a chip reset. The preserved register contents are used as the boot configuration after the chip reset.
3. Set DBCR0[RST] = 0b10 to generate a chip reset.

9.2.1.1 Booting with Option E

The following example code demonstrates how to implement the *Software Boot Configuration Procedure* for Boot Option E. When booting with Option E, the PLL is in bypass, CPR0_PLLC[ENG]=0. The *Software Boot Configuration Procedure* is needed to engage the PLL and perform a chip reset.

When SDR0_PINSTP= 0x80000000 and CPR0_PLLC[ENG]=0, the boot code must perform the *Software Boot Configuration Procedure.* The following code can be placed in the boot code prior to configuring the TLBs.

Note: Interrupts must be disabled during the *Software Boot Configuration Procedure.*

```
/* read CPR0_PLLC*/
lis r1,0x0000
ori r1,r1,0x0040
mtdcr 0xC,r1
mfdcr r2,0xD

/* mask upper bits of CPR0_PLLC*/
lis r3,0xC000
and r3,r3,r2

/*check upper bits of CPR0_PLLC to determine if engage is set */
lis r4,0x8000
cmp 0,0,r3,r4
bne noReset /* skip Software Boot Configuration Procedure if the engage bit is set */

/* set the engage bit in CPR0_PLLC*/
lis r1,0x3FFF
ori r1,r1,0xFFFF
and r2,r2,r1
lis r4,0x4000
or r2,r2,r4
lis r1,0x0000
ori r1,r1,0x0040
mtdcr 0xC,r1
mtdcr 0xD,r2

/* read CPR0_ICFG */
lis r1,0x0000
ori r1,r1,0x0140
mtdcr 0xC,r1
mfdcr r2,0xD

/* set rli bit CPR0_ICFG */
lis r4,0x8000
```

```
or r2,r2,r4
mtdcr 0xC,r1
mtdcr 0xD,r2

/*perform chip reset, DBCR0[RST]=10*/
msync
lis r4, 0x2000
mfspr r2,0x134
or r2,r2,r4
mtspr 0x134,r2
```

Table 9-2. Bootstrap Pins (Sheet 1 of 3)

UART0CTS	UART0DCD	UART0DSR	Description
0	0	0	Bootstrap Option A SysClk - 66.67 MHz VCO - 1333 MHz CPU - 667 MHz DDR - 333 MHz (data rate) PLBClk_A - 166 MHz PLBClk_B - 166 MHz OPB - 83 MHz PER - 83 MHz AHB - 166 MHz Boot ROM Location - EBC Boot width - 8 bits
0	0	1	Bootstrap Option B (Option A - High Performance) SysClk - 66.67 MHz VCO - 1600 MHz CPU - 800 MHz DDR - 400 MHz (data rate) PLBClk_A - 200 MHz PLBClk_B - 200 MHz OPB - 100 MHz PER - 100 MHz AHB - 200 MHz Boot ROM Location - EBC Boot width - 16 bits
0	1	0	Bootstrap Option C (Option A w/boot from PCI instead of Boot ROM) SysClk - 66.67 MHz VCO - 1333 MHz CPU - 667 MHz DDR - 333 MHz (data rate) PLBClk_A - 166 MHz PLBClk_B - 166 MHz OPB - 83 MHz PER - 83 MHz AHB - 166 MHz Boot ROM Location - PCI Boot width - na

Table 9-2. Bootstrap Pins (Sheet 2 of 3)

UART0CTS	UART0DCD	UART0DSR	Description
0	1	1	Bootstrap Option D (Option B w/boot from PCI instead of Boot ROM) SysClk - 66.67 MHz VCO - 1600 MHz CPU - 800 MHz DDR - 400 MHz (data rate) PLBClk_A - 200 MHz PLBClk_B - 200 MHz OPB - 100 MHz PER - 100 MHz AHB - 200 MHz Boot ROM Location - PCI Boot width - na
1	0	0	Bootstrap Option E[1, 2, 3] (Option A w/16-bit wide Boot ROM instead of 8-bit, PLL bypassed) SysClk - 66.67 MHz Clock configuration after setting CPR0_PLLC[ENG]=1 using *Software Boot Configuration Procedure*: VCO - 1333 MHz CPU - 667 MHz DDR - 333 MHz (data rate) PLBClk_A - 166 MHz PLBClk_B - 166 MHz OPB - 83 MHz PER - 83 MHz AHB - 166 MHz Clock configuration immediately after a Power-On reset: CPU - 66.67 MHz DDR - not operational, 66.67 MHz (data rata) PLBClk_A - 16.67 MHz PLBClk_B - 33.33 MHz OPB - 8.33 MHz PER - 8.33 MHz AHB - 16.67 MHz Boot ROM Location - EBC Boot width - 16 bits
1	0	1	Bootstrap Option F (Option A w/boot from NAND Flash instead of Boot ROM) SysClk - 66.67 MHz VCO - 1333 MHz CPU - 667 MHz DDR - 333 MHz (data rate) PLBClk_A - 166 MHz PLBClk_B - 166 MHz OPB - 83 MHz PER - 83 MHz AHB - 166 MHz Boot ROM Location - NAND Flash Page Size - 2KB Address Cycles - 5 cycles (2 column, 3 row) NAND Flash Bus Width - 8 bit Recommended NAND Flash Type - SLC
1	1	0	Bootstrap Option G[4, 5, 6] IIC Bootstrap controller enabled, serial ROM address 0b1010100 + r/w bit (0xA8)

Table 9-2. Bootstrap Pins (Sheet 3 of 3)

UART0CTS	UART0DCD	UART0DSR	Description
1	1	1	Bootstrap Option H[4, 5, 6] IIC Bootstrap controller enabled, serial ROM address 0b1010010 + r/w bit (0xA4)

Note:
1. When using Option E, the *Software Boot Configuration Procedure* is required.
2. JTAG debuggers — even after lowering the JTAG clock — may not function correctly when using Option E.
3. Because Option E, places the PLL in bypass mode, programming an empty boot flash using a JTAG debugger may not be possible.
4. Options G and H boot using a configuration read from an I2C serial ROM. Consider how a blank I2C serial ROM is programmed during board bring-up and manufacturing. One of the following four methods must be used:
   - Program the I2C serial ROM prior to placement on the board.
   - Program the I2C serial ROM on the board using a serial ROM programmer. This method requires a connector for access to the IIC0 bus.
   - Boot with Option A, B, C, or D and program the serial ROM using code executed from the boot flash.
   - Program the serial ROM through BSDL.
5. The serial ROM addresses for Options G and H include the R/W bit. The R/W bit is the lsb.
6. Option G and H only support serial ROM that use a one byte offset address.

Table 9-3. Bootstrap Configurations

Serial Device Strap Register Bit Field	Bootstrap Option						Description
	A	B	C	D	E	F	
SDR0_SDSTP0[ENG]	1	1	1	1	0	1	Engage 0 SysClk is the source for PLL forward dividers. 1 PLL's VCO is the source for the PLL forward dividers
SDR0_SDSTP0[SEL]	00	00	00	00	00	00	Feedback Selection 00 Local
SDR0_SDSTP0[TUNE]	11010 00000	11010 00001	11010 00000	11010 00001	11010 00000	11010 00000	PLL Tune Bits ($16 < M \leq 32$) 1101000000 (VCO = 1333 MHz) 1101000001 (VCO = 1600 MHz)
SDR0_SDSTP0[FBDV]	1011 1011	1011 1010	1011 1011	1011 1010	1011 1011	1011 1011	PLL Feedback Divisor 1011_1011 (M = 20) 1011_1010 (M = 24)
SDR0_SDSTP0[FWDVA]	0001	0001	0001	0001	0001	0001	PLL Forward Divisor A 0001 PLL Forward Divisor A = 2
SDR0_SDSTP0[FWDVB]	0100	0100	0100	0100	0100	0100	PLL Forward Divisor B 0100 PLL Forward Divisor B = 4
SDR0_SDSTP1[PLBEDV0]	100	100	100	100	100	100	PLB Early Clock Divisor 0 100 PLB Early Clock Divisor 0 = 4
SDR0_SDSTP1[PLB2DV0]	1	1	1	1	1	1	PLB2 Clock Divisor 0 1 PLB2 Clock Divisor 0 = 1
SDR0_SDSTP1[OPBDV0]	10	10	10	10	10	10	OPB Clock Divisor 0 10 OPB Clock Divisor 0 = 2
SDR0_SDSTP1[PERDV0]	01	01	01	01	01	01	Peripheral Clock Divisor 0 01 Peripheral Clock Divisor = 1

Table 9-3. Bootstrap Configurations (Continued)

Serial Device Strap Register Bit Field	Bootstrap Option						Description
	A	B	C	D	E	F	
SDR0_SDSTP1[AHBDV0]	1	1	1	1	1	1	AHB Clock Divisor 0 1 AHB Clock Divisor 0 = 1
SDR0_SDSTP1[AHBSTP]	0	0	0	0	0	0	AHB Clock Stop 0 AHB Clock Enabled
SDR0_SDSTP1[RL]	00	00	01	01	00	10	Boot ROM Location 00 EBC 01 PCI 10 NDFC
SDR0_SDSTP1[PAE]	0	0	0	0	0	0	PCI Internal Arbiter Enable 0 Disable
SDR0_SDSTP1[PHCE]	0	0	1	1	0	0	PCI Host Configuration Enable 0 Disable 1 Enable
SDR0_SDSTP1[PISE]	0	0	0	0	0	0	PCI Initial Sequence Enable 0 Disable
SDR0_SDSTP1[PCWE]	0	0	1	1	0	0	PCI Local CPU Wait Enable 0 Disable 1 Enable
SDR0_SDSTP1[PPIM]	0000	0000	0000	0000	0000	0000	PCI Inbound Map (PIM) Settings 0000 PIM0 off, PIM1 off, PIM2 off 0000 PIM0 off, PIM1 off, PIM2 off
SDR0_SDSTP1[SATA]	0	0	0	0	0	0	PE0/SATA Selection 0 PE0
SDR0_SDSTP1[SRIO]	0	0	0	0	0	0	PE1/SRIO Selection 0 PE1
SDR0_SDSTP1[RW]	00	01	00 (na)	00 (na)	01	11	Boot ROM Width 00 Boot ROM Width = 8 bit 01 Boot ROM Width = 16 bit 11 Boot ROM Width = 32 bit
SDR0_SDSTP1[RS]	00	01	00	00	01	00	Boot ROM Size 00 Boot ROM Size = 2 MB 01 Boot ROM Size = 4 MB
SDR0_SDSTP1[DBG]	0	0	0	0	0	0	Debug Mode 0 Functional
SDR0_SDSTP1[ZMIIM]	00	00	00	00	00	00	ZMII Mode 00 MII Mode
SDR0_SDSTP1[GMC1BS]	0	0	0	0	0	0	GMC Port 1 Bridge Selector 0 RGMII1 Bridge
SDR0_SDSTP1[GMC0BS]	0	0	0	0	0	0	GMC Port 0 Bridge Selector 0 RGMII0 Bridge
SDR0_SDSTP2[MEN]	01	01	01	01	01	10	Multiplex NDFC, or GPIO 01 high Z 10 NDFC

Table 9-3. Bootstrap Configurations (Continued)

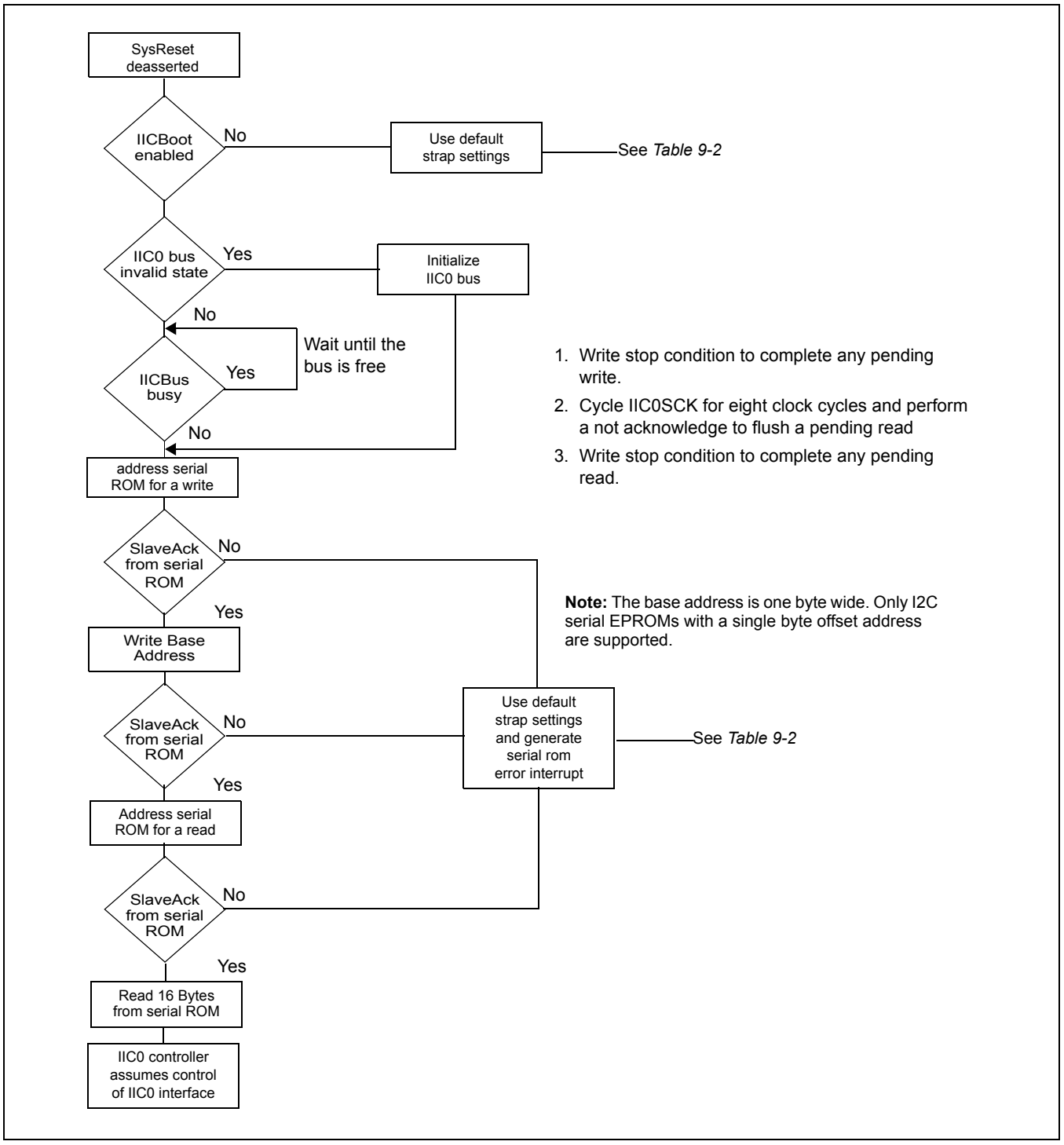
Serial Device Strap Register Bit Field	Bootstrap Option						Description
	A	B	C	D	E	F	
SDR0_SDSTP2[NE]	0	0	0	0	0	1	NDFC Enable 0 Disable 1 Enable
SDR0_SDSTP2[NBW]	0	0	0	0	0	0	NDFC Boot Width 0 8 bits
SDR0_SDSTP2[NBP]	0000	0000	0000	0000	0000	0000	NDFC Boot Page Selection
SDR0_SDSTP2[NBAC]	10	10	10	10	10	11	NDFC Boot Address Selection Cycle 10 Four cycle, 2KB per page 11 Five cycle, 2KB per page
SDR0_SDSTP2[NARE]	0	0	0	0	0	1	NDFC Auto-Read Enable 0 Disable 1 Enable
SDR0_SDSTP2[NRB]	0	0	0	0	0	0	NDFC Ready/Busy 0 Ready/Busy Disable 1 Ready/Busy Enable
SDR0_SDSTP2[NDRSC]	1000 0010 0011 0101	1000 0010 0011 0101	1000 0010 0011 0101	1000 0010 0011 0101	1000 0010 0011 0101	1000 0010 0011 0101	NDFC Device Reset Counter Value
SDR0_SDSTP2[NCG]	0000	0000	0000	0000	0000	1000	NDFC/EBC Chip Select Gating
SDR0_SDSTP3[NDRDC]	0000 1101 0000 0101	0000 1101 0000 0101	0000 1101 0000 0101	0000 1101 0000 0101	0000 1101 0000 0101	0000 1101 0000 0101	NDFC Device Read Counter Value

9.3 IIC Bootstrap Operations

After the deassertion of $\overline{\text{SysReset}}$, the IIC bootstrap controller tests the IIC bus to determine if the bus is in an invalid state or busy. If the bus is in an invalid state, it places the IIC0 bus in a known state through an initialization sequence and accesses the serial ROM as described in *Figure 9-1*. If the bus is busy, it waits until the bus is free before accessing the serial ROM. To access the ROM, the IIC bootstrap controller writes the base address (0x00) and then reads 16 bytes starting from the base address 0x00. The data in the serial ROM is organized as shown in *Table 9-4*.

If the IIC bootstrap controller is faster than the serial ROM, then the later can hold the IIC0 clock signal low until it is prepared for the next transaction. If the serial ROM does not acknowledge its address or the receipt of the base address, the PPC460EX/EXr/GT defaults to Bootstrap Option A in *Table 9-3* and generates a serial ROM error interrupt, UIC1_SR [SRE]. Once interrupts are enabled, initialization software should check UIC1_SR[SRE] to ensure no errors occurred while accessing the serial ROM.

Figure 9-1. IIC Bootstrap Controller Flow



9.3.1 Performance

The IIC bootstrap controller requires 188 IIC0_SCK clock cycles to read 16 bytes, 53 clock cycles for the first byte and 9 cycles for each of the remaining 15 bytes. The IIC0_SCK clock is generated directly from SysClk during system reset by dividing it by 768 (IIC0_SCK frequency = SysClk/768).

Assuming a SysClk of 66.66 MHz, the time needed to read the serial ROM is 2.1 mS. Serial ROM read time = 188*768/(SysClk frequency)

Table 9-4. Serial ROM Memory Map

ROM Byte Address	Bit Number Within a Byte	Register[Field]
0	7 (msb)	SDR0_SDSTP0[ENG]
	6:5	Reserved
	4:3	SDR0_SDSTP0[SEL]
	2:0 (lsb)	SDR0_SDSTP0[$TUNE_{5:7}$]
1	7:1	SDR0_SDSTP0[$TUNE_{8:14}$]
	0	Reserved
2	7:0	SDR0_SDSTP0[FBDV]
3	7:4	SDR0_SDSTP0[FBDVA]
	3:0	SDR0_SDSTP0[FBDVB]
4	7:5	SDR0_SDSTP1[PLBEDV0]
	4	SDR0_SDSTP1[PLB2DV0]
	3:2	SDR0_SDSTP1[OPBDV0]
	1:0	SDR0_SDSTP1[PERDV0]
5	7	SDR0_SDSTP1[AHBDV0]
	6	SDR0_SDSTP1[AHBSTP]
	5	Reserved
	4:3	SDR0_SDSTP1[RL]
	2	SDR0_SDSTP1[PAE]
	1	SDR0_SDSTP1[PHCE]
	0	SDR0_SDSTP1[PISE]
6	7	SDR0_SDSTP1[PCWE]
	6:3	SDR0_SDSTP1[PPIM]
	2	SDR0_SDSTP1[SATA]
	1	SDR0_SDSTP1[SRIO]
	0	SDR0_SDSTP1[RW_{23}]

Table 9-4. Serial ROM Memory Map (Continued)

ROM Byte Address	Bit Number Within a Byte	Register[Field]
7	7	SDR0_SDSTP1[RW_{24}]
	6:5	SDR0_SDSTP1[RS]
	4	SDR0_SDSTP1[DBG]
	3:2	SDR0_SDSTP1[ZMIIM]
	1	SDR0_SDSTP1[GMC1BS]
	0	SDR0_SDSTP1[GMC0BS]
8	7:6	SDR0_SDSTP2[MEN]
	5	SDR0_SDSTP2[NE]
	4	SDR0_SDSTP2[NBW]
	3:0	SDR0_SDSTP2[NBP]
9	7:6	SDR0_SDSTP2[NBAC]
	5	SDR0_SDSTP2[NARE]
	4	SDR0_SDSTP2[NRB]
	3:0	SDR0_SDSTP2[$NDRSC_{0:3}$]
A	7:0	SDR0_SDSTP2[$NDRSC_{4:11}$]
B	7:4	SDR0_SDSTP2[$NDRSC_{12:15}$]
	3:0	SDR0_SDSTP2[NCG]
C	7:0	SDR0_SDSTP3[$NDRDC_{0:7}$]
D	7:0	SDR0_SDSTP3[$NDRDC_{8:15}$]
E	7:0	Reserved
F	7:0	Reserved

9.4 PCI Bootstrap Configuration (PPC460EX and PPC460GT only)

The PCI Configuration Registers for PCI bus (SDR0_PCI0) are initialized with values stored in the SDR0_SDSTP1 for PCI0 during system reset.

For the PCI bus, there are three spaces defined for Inbound PCI-to-PLB accesses: PIM0, PIM1 and PIM2.

The PIM 0 size defined in the following table sets the low size field PCIL_PIM0SAH[31:12] = SIZL, while the PIM2 sizes sets the PCIL_PIM2SAH[31:12] = SIZL.

For the PCI bus, the PPIM field provides a selection of 16 predefined combinations of PIMs as described in *Table 9-5*.

Table 9-5. PCI Inbound Map (PIM 0, PIM 1, PIM 2) Settings

PPIM	PIM 0				PIM 1 (Size = 256 Bytes)			PIM 2			
	Enable	Prefetch	Size		Enable	Prefetch	Com-ments	Enable	Prefetch	Size	
0000	0	0	FFFFF	Off	0		Off	0	0	FFFFF	Off
0001	1	0	FFFFF	4K	0		Off	0	0	FFFFF	Off
0010	1	0	FFF00	1M	0		Off	0	0	FFFFF	Off
0011	1	0	FC000	64M	0		Off	0	0	FFFFF	Off
0100	1	1	FFFFF	4K	0		Off	0	0	FFFFF	Off
0101	1	1	FFF00	1M	0		Off	0	0	FFFFF	Off
0110	1	1	FC000	64M	0		Off	0	0	FFFFF	Off
0111	1	0	FFFF0	64K	0		Off	1	0	FFFFC	16K
1000	1	0	FFF00	1M	0		Off	1	0	FFFF0	64K
1001	1	1	FFFF0	64K	0		Off	1	0	FFFFC	16K
1010	1	1	FFF00	1M	0		Off	1	0	FFFF0	64K
1011	1	0	FFFF0	64K	0		Off	1	1	FFFF0	64K
1100	1	0	FFF00	1M	0		Off	1	1	FFF00	1M
1101	1	1	FFF00	1M	0		Off	1	1	FFF00	1M
1110	1	0	FFF00	1M	1	0	On	0	0	FFFFF	Off
1111	1	0	FFF00	1M	1	0	On	1	0	FFFFC	16K

9.5 IIC Bootstrap Registers

All bootstrap registers are accessed indirectly through the SDR0_CFGADDR and SDR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions. *Table 3-1* on page 154 lists the DCR addresses for the SDR0_CFGADDR and SDR0_CFGDATA registers.

To read or write one of the bootstrap registers, software first writes the register address offset (DCR Offset) of the target register into the SDR0_CFGADDR register. The target register can then be read or written through the SDR0_CFGDATA DCR register. The following PowerPC code illustrates this procedure by reading the SDR0_SDSTP0 register:

```
li r3, SDR0_SDSTP0_Offset
mtdcr SDR0_CFGADDR, r3      ! write SDR0_CFGADDR  with the SDR0_SDSTP0 DCR offset address
mfdcr r4, SDR0_CFGDATA      ! read the content of SDR0_SDSTP0 from SDR0_CFGDATA
```

Note: *Table 9-6* lists the device control registers that are directly affected by the bootstrap initialization process. Other system device control registers that control PPC460EX/EXr/GT miscellaneous devices are described in *Reset and Initialization* on page 283 while the clocking and power-on reset registers are described in *Clocking* on page 237.

Table 9-6. IIC Bootstrap Controller SDRs

Mnemonic	Register	DCR Offset	Access	Page
SDR0_PINSTP	Pin Strapping Register	0x0040	Read only	226
SDR0_SDCS0	Serial Device Controller Settings Register	0x0060	Read only	227
SDR0_SDSTP0	Serial Device Strap Register 0	0x0020	Read only	227
SDR0_SDSTP1	Serial Device Strap Register 1	0x0021	Read only	229
SDR0_SDSTP2	Serial Device Strap Register 2	0x0022	Read only	230
SDR0_SDSTP3	Serial Device Strap Register 3	0x0023	Read only	232
SDR0_CUST0	Customer Configuration Register 0	0x4000	R/W	232
SDR0_CUST1	Customer Configuration Register 1	0x4002	R/W	233
SDR0_DDRCE	SDRAM DDR Clock Output Enable Register	0x00E0	R/W	233
SDR0_DDRD0	SDRAM DDR Configuration Register	0x00E1	R/W	234
SDR0_EBC0	EBC Configuration Register	0x0100	R/W	234
SDR0_PCI0	PCI Configuration Register	0x01C0	R/W	235
SDR0_PFC1	Pin Function Control Register 1	0x4101	R/W	236

9.5.1 Pin Strapping Register (SDR0_PINSTP)

The Pin Strapping Register (SDR0_PINSTP) records how the bootstrap pins are strapped during system reset. A copy of pin strap status is also maintained by *Initial Configuration Register (CPR0_ICFG)* on page 251.

Figure 9-2. Pin Strapping Register (SDR0_PINSTP)

0	UCTS	$\overline{\text{UART0CTS}}$ Strapping 0 Strapped low 1 Strapped high	
1	UDCD	$\overline{\text{UART0DCD}}$ Strapping 0 Strapped low 1 Strapped high	

2	UDSR	$\overline{\text{UART0DSR}}$ Strapping 0 Strapped low 1 Strapped high	
3:31		Reserved	

9.5.2 Serial Device Controller Settings Register (SDR0_SDCS0)

The Serial Device Controller Settings Register contains the serial ROM address and status showing if the IIC Bootstrap Controller was enabled during system reset.

Figure 9-3. Serial Device Controller Settings Register (SDR0_SDCS0)

0:6	SBA	Serial ROM Base Address: $\overline{\text{UART0DSR}}$ = 0 Base address 0xA8 0b1010100 $\overline{\text{UART0DSR}}$ = 1 Base address 0xA4 0b1010010	
7:29		Reserved	
30	SDC	IIC Serial Device Control 0 Serial device control disabled 1 Serial device control enabled	Status bit indicating whether the Serial Device Controller was enabled during the most recent system reset, and attempted to read a serial device. This bit is often controlled by a pin strap that is also recorded in the SDR0_PINSTP register.
31	SDD	Serial Device Detection 0 Serial device detection disabled 1 Serial device detection enabled	Status bit indicating whether a Serial ROM device was detected during the most recent system reset. This bit is set to 1 only if the Serial Device Controller was enabled and able to read a serial device successfully during the most recent system reset.

9.5.3 Serial Device Strap Register 0 (SDR0_SDSTP0)

SDR0_SDSTP0 is 32-bit read-only register. This register is reserved for system PLL and configuration information. The SDR0_SDSTP0 is reset according to values read by the IIC Bootstrap controller or provided by bootstrap options described in *Bootstrap Options* on page 215. If the IIC Bootstrap controller is unable to read the boot-strap information from a serial ROM device, the bootstrap option A is used as the default configuration.

Figure 9-4. Serial Device Strap Register 0 (SDR0_SDSTP0)

0	ENG	Engage 0 SysClk is the source for PLL forward dividers. 1 CPU PLL's VCO is the source for PLL forward dividers.	Set SYS_PLL Control Register (CPR0_PLLC [ENG])
1:2		Reserved	
3:4	SEL	CPU PLL Feedback Selection 00 Local 11 PerClk	Set SYS_PLL Control Register (CPR0_PLLC [SEL])
5:14	TUNE	CPU PLL TUNE bits	Set SYS_PLL Control Register (CPR0_PLLC [TUNE])
15		Reserved	

16:23	FBDV	CPU PLL Feedback Divisor 0000_0000 PLL Feedback Divisor = 1 1111_1111 PLL Feedback Divisor = 2 0111_1110 PLL Feedback Divisor = 3 1111_1101 PLL Feedback Divisor = 4 0111_1010 PLL Feedback Divisor = 5 1111_0101 PLL Feedback Divisor = 6 0110_1010 PLL Feedback Divisor = 7 1101_0101 PLL Feedback Divisor = 8 .... 1001_1111 PLL Feedback Divisor = 254 0011_1111 PLL Feedback Divisor = 255	Set SYS_PLL Divisor Register (CPR0_PLLD [FBDV])
24:27	FWDVA	CPU PLL Forward Divisor A 0000 PLL Forward Divisor A = 1 0001 PLL Forward Divisor A = 2 1111 PLL Forward Divisor A = 3 0100 PLL Forward Divisor A = 4 1001 PLL Forward Divisor A = 5 1010 PLL Forward Divisor A = 6 1101 PLL Forward Divisor A = 7 1110 PLL Forward Divisor A = 8 0011 PLL Forward Divisor A = 9 1100 PLL Forward Divisor A = 10 0101 PLL Forward Divisor A = 11 1000 PLL Forward Divisor A = 12 0111 PLL Forward Divisor A = 13 0010 PLL Forward Divisor A = 14 1011 PLL Forward Divisor A = 15 0110 PLL Forward Divisor A = 16	Set SYS_PLL Divisor Register (CPR0_PLLD [FWDVA])
28:31	FWDVB	CPU PLL Forward Divisor B 0000 PLL Forward Divisor A = 1 0001 PLL Forward Divisor A = 2 1111 PLL Forward Divisor A = 3 0100 PLL Forward Divisor A = 4 1001 PLL Forward Divisor A = 5 1010 PLL Forward Divisor A = 6 1101 PLL Forward Divisor A = 7 1110 PLL Forward Divisor A = 8 0011 PLL Forward Divisor A = 9 1100 PLL Forward Divisor A = 10 0101 PLL Forward Divisor A = 11 1000 PLL Forward Divisor A = 12 0111 PLL Forward Divisor A = 13 0010 PLL Forward Divisor A = 14 1011 PLL Forward Divisor A = 15 0110 PLL Forward Divisor A = 16	Set SYS_PLL Divisor Register (CPR0_PLLD [FWDVB])

9.5.4 Serial Device Strap Register 1 (SDR0_SDSTP1)

SDR0_SDSTP1 is 32-bit read-only register. This register is reserved for system PLL and configuration information. The SDR0_SDSTP1 is reset according to values read by the IIC Bootstrap controller or provided by bootstrap options described in *Bootstrap Options* on page 215. If the IIC Bootstrap controller is unable to read the boot-strap information from a serial ROM device, the bootstrap option A is used as the default configuration

Figure 9-5. Serial Device Strap Register 1 (SDR0_SDSTP1)

0:2	PLBEDV0	PLB Early Clock Divisor 0 000 PLB Early clock divisor 0 = 8 001 PLB Early clock divisor 0 = 1 010 PLB Early clock divisor 0 = 2 011 PLB Early clock divisor 0 = 3 100 PLB Early clock divisor 0 = 4 101 PLB Early clock divisor 0 = 5 110 PLB Early clock divisor 0 = 6 111 PLB Early clock divisor 0 = 7	Set PLB Early Clock Divisor Register (CPR0_PLBED [PLBEDV0])
3	PLB2DV0	PLB2 Clock Divisor 0 0 PLB2 clock divisor 0 = 2 1 PLB2 clock divisor 0 = 1	Set PLB2 Clock Divisor Register (CPR0_PLB2D [PLB2DV0])
4:5	OPBDV0	OPB Clock Divisor 0 00 OPB clock divisor 0 = 4 01 OPB clock divisor 0 = 1 10 OPB clock divisor 0 = 2 11 OPB clock divisor 0 = 3	Set OPB Clock Divisor Register (CPR0_OPBD [OPBDV0])
6:7	PERDV0	Peripheral Clock Divisor 0 00 Peripheral clock divisor 0 = 4 01 Peripheral clock divisor 0 = 1 10 Peripheral clock divisor 0 = 2 11 Peripheral clock divisor 0 = 3	Set Peripheral Clock Divisor Register (CPR0_PERD [PERDV0])
8	AHBDV0	AHB Clock Divisor 0 0 AHB clock divisor 0 = 2 1 AHB clock divisor 0 = 1	Set AHB Clock Divisor Register (CPR0_AHBD [AHBDV0])
9	AHBSTP	AHB Clock Stop 0 AHB Clock enabled 1 AHB Clock disabled	Set AHB Subsystem Configuration Register (SDR0_AHB_CFG [AHBSTP])
10		Reserved	
11:12	RL	ROM Location 00 EBC 01 PCI 10 NDFC 11 Reserved	Set PPC440 CPU Register (SDR0_CP440 [RL])
13	PAE	PCI Arbiter Enable: 0 PCI arbiter disabled 1 PCI arbiter enabled	Set PCI Configuration Register (SDR0_PCI0 [PAE])
14	PHCE	Enable PCI to be Configured by External Host: 0 PCI host configuration disabled 1 PCI host configuration enabled	Set PCI Configuration Register (SDR0_PCI0 [PHCE])
15	PISE	PCI Initial Sequence Mode Definition: 0 Adapter mode 1 Host mode	Set PCI Configuration Register (SDR0_PCI0 [PISE])
16	PCWE	PCI Local CPU Wait Enable: 0 PCI local CPU wait disabled 1 PCI local CPU wait enabled	Set PCI Configuration Register (SDR0_PCI0 [PCWE])

17:20	PPIM	PCI Inbound Map (PIM) Settings 0000 PIM0 off, PIM1 off, PIM2 off 0001 PIM0 4K, PIM1 off, PIM2 off 0010 PIM0 1M, PIM1 off, PIM2 off 0011 PIM0 64M, PIM1 off, PIM2 off 0100 PIM0 4K prefetch enabled, PIM1 off, PIM2 off 0101 PIM0 1M prefetch enabled, PIM1 off, PIM2 off 0110 PIM0 64M prefetch enabled, PIM1 off, PIM2 off 0111 PIM0 64K, PIM1 off, PIM2 16K 1000 PIM0 1M, PIM1 off, PIM2 64K 1001 PIM0 64K prefetch enabled, PIM1 off, PIM2 16K 1010 PIM0 1M prefetch enabled, PIM1 off, PIM2 64K 1011 PIM0 64K, PIM1 off, PIM2 64K prefetch enabled 1100 PIM0 1M, PIM1 off, PIM2 1M prefetch enabled 1101 PIM0 1M prefetch enabled, PIM1 off, PIM2 1M prefetch enabled 1110 PIM0 1M, PIM1 on, PIM2 off 1111 PIM0 1M, PIM1 on, PIM2 16K	Set PCI Configuration Register (SDR0_PCI0 [PPIM])
21	SATA	PE0 / SATA selection 0 PE0 1 SATA	Set PCI Express 0 PHY Control Reset Register (SDR0_PE0_PHY_CTL_RST [31])
22	SRIO	PE1 / SRIO selection 0 PE1 1 SRIO	Set PCI Express 1 PHY Control Reset Register (SDR0_PE1_PHY_CTL_RST [30])
23:24	RW	ROM Width 00 8-bit ROM 01 16-bit ROM 10 Reserved 11 32-bit ROM (Mandatory for Boot from Nand Flash)	Set EBC Configuration Register (SDR0_EBC0 [RW])
25:26	RS	ROM Size 00 2MB 01 4MB 10 8MB 11 16MB	Set EBC Configuration Register (SDR0_EBC0 [RS]) + 0b001
27	DBG	Debug Mode 0 Functional 1 CPU trace	Set Pin Function Control Register 0 (SDR0_PFC0 [DBG])
28:29	ZMIIM	ZMII bridge mode selector 00 MII 01 Reserved 10 Reserved 11 Reserved	Set Ethernet Configuration Register (SDR0_ETH_CFG [ZMIIM])
30	GMC1BS	GMC Port 1 bridge selector 0 RGMII1 Bridge 1 Reserved	Set Ethernet Configuration Register (SDR0_ETH_CFG [GMC1BS])
31	GMC0BS	GMC Port 0 bridge selector 0 RGMII0 Bridge 1 ZMII Bridge	Set Ethernet Configuration Register (SDR0_ETH_CFG [GMC0BS])

9.5.5 Serial Device Strap Register 2 (SDR0_SDSTP2)

SDR0_SDSTP2 is a 32-bit read-only register. This register is reserved for user defined initialization data. The SDR0_SDSTP2 is reset according to values read by the IIC Bootstrap controller or provided by bootstrap options described in *Bootstrap Options* on page 215. If the IIC Bootstrap controller is unable to read the boot-strap information from a serial ROM device, the bootstrap option A is used as the default configuration.

Figure 9-6. Serial Device Strap Register 2 (SDR0_SDSTP2)			
0:1	MEN	Multiplex NDFC or GPIO 00 Reserved 01 Reserved 10 NDFC 11 GPIO	[GPIO22]$\overline{\text{NFRdyBusy}}$ [GPIO23]$\overline{\text{NFREn}}$ [GPIO24]$\overline{\text{NFWEn}}$ [GPIO25]NFCLE [GPIO26]NFALE Set Customer Configuration Register 0 (SDR0_CUST0[MEN])
2	NE	NDFC Enable 0 Disabled 1 Enabled	Set Customer Configuration Register 0 (SDR0_CUST0[NE])
3	NBW	NDFC Boot Width 0 8-bit 1 16-bit	Width of external device. Set Customer Configuration Register 0 (SDR0_CUST0[NBW])
4:7	NBP	NDFC Boot Page selection	Typically, page 0 is selected. Set Customer Configuration Register 0 (SDR0_CUST0[NBP])
8:9	NBAC	NDFC Boot Address Selection Cycle 00 3 cycles, 1 Col. + 2 Row (512B page) 01 4 cycles, 1 Col. + 3 Row (512B page) 10 4 cycles, 2 Col. + 2 Row (2KB page) 11 5 cycles, 2 Col. + 3 Row (2KB page)	Set Customer Configuration Register 0 (SDR0_CUST0[NBAC])
10	NARE	NDFC Auto-Read Enable 0 Disabled 1 Enabled	When booting from NAND flash, set NARE = 1. Set Customer Configuration Register 0 (SDR0_CUST0[NARE])
11	NRB	NDFC Ready/Busy 0 Ready/Busy disable 1 Ready/Busy enable	When booting from NAND flash, set NRB = 0. When NRB = 1, the NDFC waits for NFRdyBusy to be sampled high before allowing read cycles to occur. Set Customer Configuration Register 0 (SDR0_CUST0[NRB])
12:27	NDRSC	NDFC Device Reset Counter value Number PerClk cycles NDFC waits for I/O signal NFRdyBusy = 1 after reset.	Many NAND flash devices specify a worst-case reset time of 500µs. **Example:** When PerClk = 33.33MHz (30ns) then NDRSC = 0x411B = 16667 cycles = 500µs Set Customer Configuration Register 0 (SDR0_CUST0[NDRSC])
28	NCG0	NDFC/EBC Chip select (enable) gating Bit 28: CS0 (0 = EBC, 1 = NDFC)	Select either the EBC or the NDFC to control chip select (enable) pins. $\overline{\text{PerCSn}}$ (chip select) for EBC or $\overline{\text{NFCEn}}$ (chip enable) for NDFC Set Customer Configuration Register 0 (SDR0_CUST0[NCG0-3])
29	NCG1	NDFC/EBC Chip select (enable) gating Bit 29: CS1 (0 = EBC, 1 = NDFC)	
30	NCG2	NDFC/EBC Chip select (enable) gating Bit 30: CS2 (0 = EBC, 1 = NDFC)	
31	NCG3	NDFC/EBC Chip select (enable) gating Bit 31: CS3 (0 = EBC, 1 = NDFC)	

9.5.6 Serial Device Strap Register 3 (SDR0_SDSTP3)

SDR0_SDSTP3 is a 32-bit read-only register. This register is reserved for user defined initialization data. The SDR0_SDSTP3 is reset according to values read by the IIC Bootstrap controller or provided by bootstrap options described in *Bootstrap Options* on page 215. If the IIC Bootstrap controller is unable to read the boot-strap information from a serial ROM device, the bootstrap option A is used as the default configuration.

Figure 9-7. Serial Device Strap Register 3 (SDR0_SDSTP3)

0:15	NDRSC	Device Read Count Maximum number of PerClk cycles NDFC waits for NFRdyBusy = 1.	Set NDRSC to allow 50μs for a page read command to complete. When NFRdyBusy goes high before reaching the NDRSC count, the counter resets. The EBC transfer wait state timer (EBC0_B0AP[TWT]) starts counting wait states when either NFRdyBusy goes high or the count exceeds NDRSC. **Example:** For PerClk = 33.33MHz (30ns), set NDRSC = 0x0683 = 1667 cycles = 50μs Set Customer Configuration Register 1 (SDR0_CUST1[NDRSC])
16:31		Reserved	

9.5.7 Customer Configuration Register 0 (SDR0_CUST0)

The Custom Configuration Register 0 (SDR0_CUST0) contains user defined initialization data. The bootstrap controller reads 8 bytes of user defined bootstraps and stores them in the SDR0_SDSTP2 register. The contents of read-only register SDR0_SDSTP2 initializes SDR0_CUST0.

Figure 9-8. Serial Device Strap Register 0 (SDR0_CUST0)

0:1	MEN	Multiplex NAND Flash or GPIO 00 Reserved 01 Reserved 10 NDFC 11 GPIO	
2	NE	NDFC Enable 0 Disabled 1 Enabled	
3	NBW	NDFC Boot Width 0 8-bit 1 16-bit	Width of external device.
4:7	NBP	NDFC Boot Page Selection	Typically, page 0 is selected.
8:9	NBAC	NDFC Boot Address Select Cycle 00 3 Addr. Cycles, 1 Col. + 2 Row (512 page size) 01 4 Addr. Cycles, 1 Col. + 3 Row (512 page size) 10 4 Addr. Cycles, 2 Col. + 2 Row (2k page size) 11 5 Addr. Cycles, 2 Col. + 3 Row (2k page size)	
10	NARE	NDFC Auto Read Disable 0 Disabled 1 Enabled	When booting from NAND flash, set NARE = 1.
11	NRB	NDFC Ready/Busy 0 Ready/Busy disable 1 Ready/Busy enable	When booting from NAND flash, set NRB = 0. When NRB = 1, the NDFC waits for NFRdyBusy to be sampled high before allowing read cycles to occur.

12:27	NDRSC	NDFC Device Reset Counter Value	Many NAND flash devices specify a worst-case reset time of 500µs. **Example:** When PerClk = 33.33MHz (30ns) then NDRSC = 0x411B = 16667 cycles = 500µs
28	NCG0	NDFC/EBC Chip select (enable) gating Bit 28: CS0 (0 = EBC, 1 = NDFC)	Select either the EBC or the NDFC to control chip select (enable) pins. **Note:** PerCSn (chip select) for EBC or NFCEn (chip enable) for NDFC
29	NCG1	NDFC/EBC Chip select (enable) gating Bit 29: CS1 (0 = EBC, 1 = NDFC)	
30	NCG2	NDFC/EBC Chip select (enable) gating Bit 30: CS2 (0 = EBC, 1 = NDFC)	
31	NCG3	NDFC/EBC Chip select (enable) gating Bit 31: CS3 (0 = EBC, 1 = NDFC)	

9.5.8 Custom Configuration Register 1 (SDR0_CUST1)

The Custom Configuration Register (SDR0_CUST1) contains user defined initialization data. The bootstrap controller reads bytes of user defined bootstraps and stores them in the SDR0_SDSTP3 register. The contents of read-only register SDR0_SDSTP3 initializes SDR0_CUST1.

Figure 9-9. Custom Configuration Register 1 (SDR0_CUST1)

0:15	NDRSC	Device Read Count	Set NDRSC to allow 50µs for a page read command to complete. When NFRdyBusy goes high before reaching the NDRSC count, the counter resets. The EBC transfer wait state timer (EBC0_B0AP[TWT]) starts counting wait states when either NFRdyBusy goes high or the count exceeds NDRSC. **Example:** For PerClk = 33.33MHz (30ns), set NDRSC = 0x0683 = 1667 cycles = 50µs
16:31		Reserved	Reserved for user's data

9.5.9 DDR Clock Output Enable Register (SDR0_DDRCE)

SDR0_DDRCE enables the output clock signals generated by the PPC460EX/EXr/GT for attached memories.

Figure 9-10. DDR Clock Output Enable Register (SDR0_DDRCE)

0:2		Reserved	
3		MemClkOut0 SDRAM DDR Clock Signal Enable	Reset value = 1 (enable).
4:6		Reserved	
7		MemClkOut1 SDRAM DDR Clock Signal Enable	Reset value = 1 (enable).
8:31		Reserved	

9.5.10 SDRAM DDR Configuration Register (SDR0_DDRD0)

SDR0_DDRD0 defines the settings of the DDR to PLB clock ratio.

Figure 9-11. SDRAM DDR Configuration Register (SDR0_DDRD0)

0		Reserved	
1:2	DDRM	DDR:PLB Clock ratio 00 Reserved 01 DDR:PLB ratio 1:1 10 DDR:PLB ratio 2:1 11 Reserved	Reset value = 0b01 (1:1).
3:31		Reserved	

9.5.11 EBC Configuration Register (SDR0_EBC0)

The EBC Configuration Register (SDR0_EBC0) is initialized with values provided by the SDR0_SDSTP1[RW] and SDR0_SDSTP1[RS] bit field. This register configures EBC bank 0 peripheral width and size when booting from an EBC attached ROM.

Figure 9-12. EBC Configuration Register (SDR0_EBC0)

0:1		Reserved	
2:3	RW	ROM Width 00 8-bit ROM 01 16-bit ROM 10 Reserved 11 32-bit ROM (mandatory for boot from Nand Flash)	When booting from NDFC, this value should be 11, as in boot configuration F. Since the actual device is hidden from the EBC, 32 bits indicates the width of the controller itself. Reset value = SDR0_SDSTP1[RW]
4:16		Reserved	
17:19	RS	ROM Size 001 2MB 010 4MB 011 8MB 100 16MB	Reset value = SDR0_SDSTP1[RS] + 0b001
20:31	RBA	ROM Base Address per ROM Size 0xFFE for 2MB 0xFFC for 4MB 0xFF8 for 8MB 0xFF0 for 16MB	The 12 most significant bits of the 32-bit ROM base address

9.5.12 PCI Configuration Register (SDR0_PCI0)

SDR0_PCI0 contains PCI configuration information. It is configured by bootstraps. See *Table 9-1* on page 213 for more information.

Figure 9-13. PCI Configuration Register (SDR0_PCI0)

0	PAE	PCI Arbiter Enable: 0 PCI arbiter disabled 1 PCI arbiter enabled	Reset value = SDR0_SDSTP1[PAE]
1	PHCE	Enable PCI to be Configured by External Host: 0 PCI host configuration disabled 1 PCI host configuration enabled	Reset value = SDR0_SDSTP1[PHCE]
2	PISE	PCI Initial Sequence Mode Definition: 0 Adapter mode 1 Host mode	Reset value = SDR0_SDSTP1[PISE]
3	PCWE	PCI Local CPU Wait Enable: 0 PCI local CPU wait disabled 1 PCI local CPU wait enabled	Reset value = SDR0_SDSTP1[PCWE]
4:7	PPIM	PCI Inbound Map (PIM) Settings 0000 PIM0 off, PIM1 off, PIM2 off 0001 PIM0 4K, PIM1 off, PIM2 off 0010 PIM0 1M, PIM1 off, PIM2 off 0011 PIM0 64M, PIM1 off, PIM2 off 0100 PIM0 4K prefetch enabled, PIM1 off, PIM2 off 0101 PIM0 1M prefetch enabled, PIM1 off, PIM2 off 0110 PIM0 64M prefetch enabled, PIM1 off, PIM2 off 0111 PIM0 64K, PIM1 off, PIM2 16K 1000 PIM0 1M, PIM1 off, PIM2 64K 1001 PIM0 64K prefetch enabled, PIM1 off, PIM2 16K 1010 PIM0 1M prefetch enabled, PIM1 off, PIM2 64K 1011 PIM0 64K, PIM1 off, PIM2 64K prefetch enabled 1100 PIM0 1M, PIM1 off, PIM2 1M prefetch enabled 1101 PIM0 1M prefetch enabled, PIM1 off, PIM2 1M prefetch enabled 1110 PIM0 1M, PIM1 on, PIM2 off 1111 PIM0 1M, PIM1 on, PIM2 16K	These bits control the default settings for various PIM fields. See *Table 9-5* on page 225 for details. Reset value = SDR0_SDSTP1[PPIM]
8:31		Reserved	

9.5.13 Pin Function Control Register 1 (SDR0_PFC1)

Figure 9-14. Pin Function Control Register 1 (SDR0_PFC1)			
0:5		Reserved	
6	U1ME	UART1 Mode Enable Selects the function of UART1 signals in 4-pin mode. 0 Enable UART1DSR/CTS as DSR and UART1DTR/RTS as DTR 1 Enable UART1DSR/CTS as CTS and UART1DTR/RTS as RTS	See multiplexed signals GPIO34 and GPIO35 in the PPC460EX/EXr/GT data sheet. **Note:** U0IM must be set to 1 for 4-pin mode.
7:11		Reserved	
12	U0ME	UART0 Mode Enable Selects the function of UART0 signals in 4-pin mode. 0 Enable UART0DSR/CTS as DSR and UART0DTR/RTS as DTR 1 Enable UART0DSR/CTS as CTS and UART0DTR/RTS as RTS	See multiplexed signals GPIO36 and GPIO37 in the PPC460EX/EXr/GT data sheet. **Note:** U0ME must be set to 1 when U0IM = 0.
13	U0IM	UART0 Interface Mode 0 UART0 interface mode has 8 pins 1 UART0 interface mode has 4 pins	
14	SIS	SPI/IIC1 Selection 0 SPI enabled 1 IIC1 enabled	SIS controls the function of signals: [IIC1SClk]SCPClkOut [IIC1SData]SCPDI
15:31		Reserved	

10. Clocking

Clocking is highly configurable and supports a wide range of clock ratios on the internal and external buses. Clocking and power-on reset (CPR) registers enable flexible power-on configuration using the IIC bootstrap controller, as well as changes after the PPC460EX/EXr/GT has begun operation. Maximum performance at different operating frequencies is possible through the support of integral relative frequencies for CPU and PLB clocks, including ratios such as 2:1, 3:1, 4:1, 5:1, 6:1, 7:1, and 8:1. Examples of clock operating points are shown in *Table 10-1*.

The PPC460EX/EXr/GT uses two PLLs to support the System and Ethernet clocking:

- SYS_PLL - source for CPU and PLB2X/DDR2X clocks and, indirectly, the PLB/DDR1X, OPB, AHB, serial, and external bus clocks
- Ethernet PLL - source for Ethernet clock

The control and configuration necessary for each PLL is described in this chapter. Ensure that you properly filter both the analog VDD and GND inputs used by each PLL (described in detail in the PPC460EX/EXr/GT data sheets).

PLL operation is controlled by a set of registers defined in *System (CPU) Clocking Registers* on page 245. The Voltage Controlled Oscillator (VCO) contained within the PLLs is required to operate in the range from 600-2000 MHz. *System Clock Ratios* on page 243 describes the VCO, CPU, PLB, OPB, and PER (EBC) frequencies set by the clocking registers.

10.1 Examples of System Clocking Operating Points

The following table illustrates different operating points for the CPU, DDR, PLB, and OPB. The table assumes a SysClk frequency of 66.67 MHz with local feedback. The M value is equal to the feedback divider (FBDV) with values in the range from 18 to 30. This M value range is based on the selecting the minimum jitter for VCO range from the PLL specification. The PLL uses local feed back. The DDR:PLB ratio is 1:1.

Table 10-1. Clock Configuration

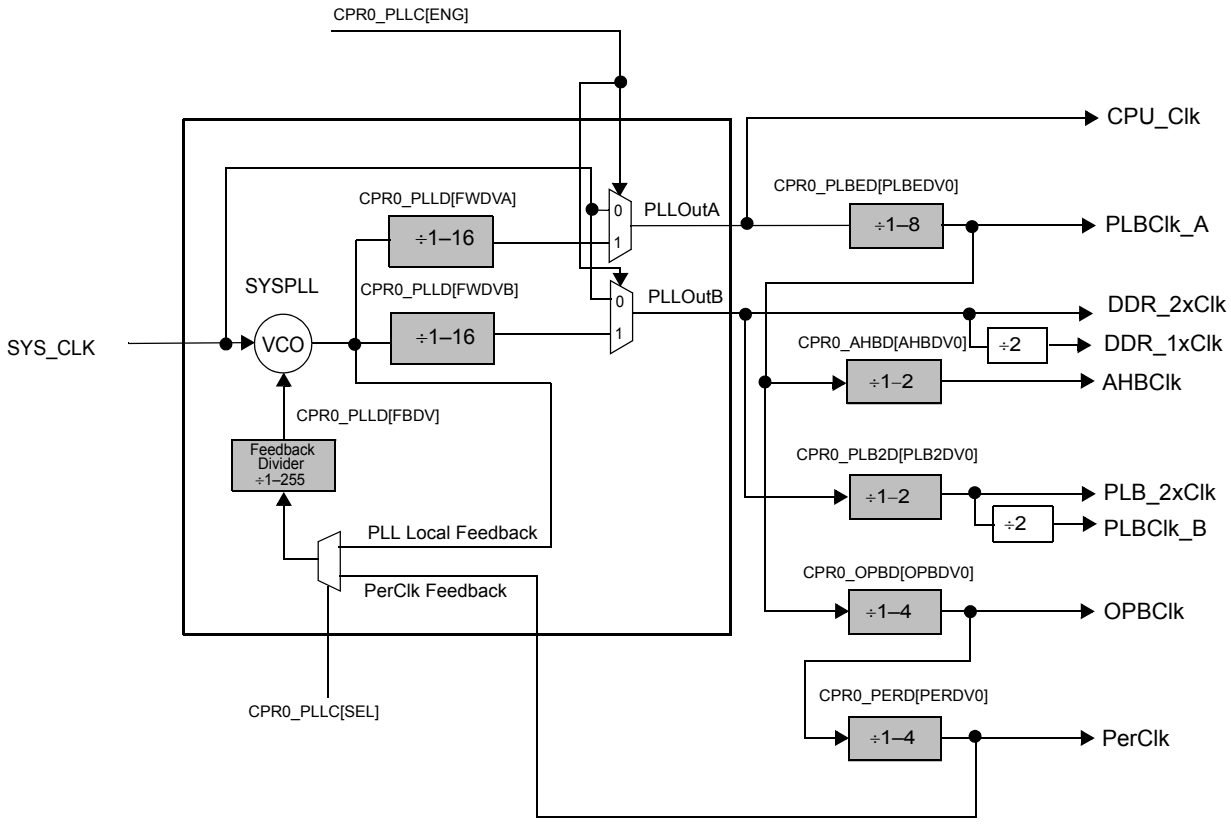
CPUClk (MHz)	PLBClk (MHz)	DDR1xClk (MHz)	VCO (MHz)	FWDVA	FWDVB	FBDV	PLBEDV0	PLB2DV0
400.00	200.00	200.00	800.00	2	1	12	1	1
533.33	133.33	133.33	1066.66	2	4	16	4	1
533.33	177.77	177.77	1066.66	2	3	16	3	1
600.00	200.00	200.00	1200.00	2	3	18	3	1
666.67	133.33	133.33	1333.33	2	5	20	5	1
666.67	166.67	166.67	1333.33	2	4	20	4	1
800.00	133.33	133.33	1600.00	2	6	24	6	1
800.00	160.00	160.00	1600.00	2	5	24	5	1
800.00	200.00	200.00	1600.00	2	4	24	4	1
833.33	166.67	166.67	1666.66	2	5	25	5	1
933.33	133.33	133.33	1866.66	2	7	28	7	1
933.33	155.55	155.55	1866.66	2	6	28	6	1
933.33	186.66	186.66	1866.66	2	5	28	5	1
1000.00	142.86	142.86	2000.00	2	7	30	7	1
1000.00	166.67	166.67	2000.00	2	6	30	6	1
1000.00	200.00	200.00	2000.00	2	5	30	5	1
1066.66	133.33	133.33	1066.66	1	4	16	8	1
1066.66	177.77	177.77	1066.66	1	3	16	6	1

CPR0_PLLD[FWDVA] -PLL Forward Divisor A
CPR0_PLLD[FWDVB] - PLL Forward Divisor B
CPR0_PLLD[FBDV] - PLL Feedback Divisor
CPR0_PLBED[PLBEDV0] - PLB Early Clock Divisor 0
CPR0_PLBD[PLB2DV0] - PLB Clock Divisor 0
For 1:1 DDR:PLB ratio, set CPR0_PLB2D[PLB2DV0] = divide by 1 and SDR0_DDR0[DDRM] = 1:1 ratio. SDR0_DDR0[DDRM] by default is set to 1:1 DDR:PLB ratio.

10.2 System Clocking

Figure 10-1 shows the clocks related to general system operation of chip. Note that separate copies of PLB clock, PLBClk_A and PLBClk_B, are generated from the PLL outputs, PLLOutA and PLLOutB, respectively. Both of these clocks must be programmed to the same frequency.

Figure 10-1. PPC460EX/EXr/GT System Clocking



Note:

1. Serial clock for the UARTs is derived internally from PLBClk or externally from UARTSerClk. See *Section 31.2 Serial Port Clocking* on page 1090.
2. The time base for the PPC440 CPU Timer Facilities is derived either internally from CPUClk or externally from TmrClk. For clock sources, see *Section 10.2.6 Source Clocks for UART, Timers, MDIO, IIC and SPI* on page 242.
3. PerClk feedback is not recommended.
4. CPR0_PLB2D[PLB2DV0] must be set to divide by 1 and SDR0_DDR0[DDRM] must be set to 1:1 ratio.
5. PLBClk_A and PLBClk_B must be the same frequency.

10.2.1 Input System Clock

The system clock (SysClk) provided to the PPC460EX/EXr/GT must be in the range 66.67 MHz to 100 MHz. It must be monotonic and have acceptable jitter and slew rates as specified in the *PPC460EX/EXr/GT Data Sheet*.

10.2.2 Feedback Selection

The PPC460EX/EXr/GT clocking logic operates in three different modes: PLL bypass mode, PLL engaged with PLL local feedback, and PLL engaged with PerClk feedback as shown in the following table:

Table 10-2. System PLL Feedback Selection

CPR0_PLLC[ENG]	CPR0_PLLC[SEL]	Mode
0	xx	Bypass
1	00	PLL Local Feedback
1	11	PerClk Feedback
Note: PerClk feedback is not recommended.		

The PLL operates in Bypass mode upon entering Reset (see *Reset and Initialization* on page 283 for additional information about reset). In this mode the system reference clock, SysClk, replaces the output of the PLL. The frequencies of the various clocks are SysClk divided by all the divisors in that clock's path. The PLL remains in Bypass mode until the initial configuration is determined from either the default serial ROM straps or the values read in from the serial ROM device. When the initial configuration has been determined, set up, and the clocks are stable, reset is released to the rest of the PPC460EX/EXr/GT.

The PPC460EX/EXr/GT system clocking logic is designed to use one of two feedback paths for the SYS_PLL, selected by CPR0_PLLC[SEL]. Most users will find that PLL local feedback provides the greatest flexibility in choosing CPU and PLB clock ratios. Users who require that the external bus clock, PerClk, be both frequency- and phase-aligned with SysClk will need to use PerClk for feedback.

For PerClk and SysClk to be phase aligned, the clock dividers must be selected such that the frequencies are the same. Doing this requires that the CPR0_PLLD[FBDV] bits be set to divide-by-one, and that PLB and other clocks derived from PLLOUTA are a multiple of PerClk. Note the supported PerClk frequency range in PerClk feedback mode is the same as the range for SysClk. In the absence of any system requirement for phase alignment, between PerClk and SysClk, users are encouraged to use PLL local feedback mode.

Changing clocking modes can be done by programming the clocking control registers to the desired values for the new clocking mode, enabling Reload Inhibit (CPR0_ICFG[RLI] =1), and performing a chip reset (DBCR0[RST] = 0b10). See *Bootstrap Options* on page 215 for instructions on changing clocking modes through software.

10.2.3 VCO Frequency and M Value for SYS_PLL

For any acceptable input SysClk frequency, the SYS_PLL VCO frequency is set by the total product of divider circuits used in the path from VCO output back to VCO input, multiplied by the system clock frequency. The product of the divider circuits, both inside the PLL and in the external divider circuits, such as those that generate the PLB, OPB, and PerClk, is referred to as the M multiplier value. Knowing this value is required in order to set the TUNE bits correctly.

The M value for the SYS_PLL can be calculated using any of the equations shown in the following table, depending upon the choice made for feedback as shown in the subsequent tables.

Table 10-3. Clock Frequencies and M Value

Feedback Choice	M Multiplier Equation
PLL Local	M = FBDV
PerClk Clock	M = FWDVA x PLBEDV0 x OPBDV0 x PERDV0 (FBDV = 1) **Note:** PerClk feedback is not recommended.

As an example of the effect various settings have on clock frequencies and the M value, consider the System PLL configured with PLL Local feedback and with PerClk feedback as shown in the following table.

Table 10-4. System PLL Configuration and Feedback Selection

PLL common Settings	PLL Local Feedback	PerClk Feedback[1]
CPR0_PLLC[SEL]	00	11
SYSCLK = 66.67MHz	VCO = 1333MHz	VCO = 1333MHz
FWDVA = 2	CPU = 667MHz	CPU = 667MHz
FWDVB = 4, DDRDV0 = 1	DDR2X/DDR1X = 333/166MHz	DDR2X/DDR1X = 333/166MHz
PLBEDV0 = 1	PLB2X/PLB1X = 333/166MHz	PLB2X/PLB1X = 333/166MHz
OPBDV0 = 2	OPB = 66.6MHz	OPB = 100MHz
PERDV0 = 1	PerClk = 66.6MHz	PerClk = 33.3MHz
M= 20	FBDV = 20	FBDV = 1
TUNE	TUNE = 1101000000	TUNE = 1101000000
1. **Note:** PerClk feedback is not recommended.		

10.2.4 SYS_PLL TUNE Setting

The SYS_PLL must be provided with different TUNE bit settings based upon the VCO frequency and M value, both of which result from the system configuration. *Table 10-5* describes how to determine the proper CPR0_PLLC[TUNE] bit settings for appropriate application.

Table 10-5. CPR0_PLLC[TUNE] Bit Settings

M Value Range	VCO Frequency Range (MHz)	TUNE									
		9	8	7	6	5	4	3	2	1	0
$10 < M \leq 16$	$600 \leq VCO < 700$	1	1	0	0	0	0	0	0	0	0
$10 < M \leq 16$	$700 \leq VCO < 800$	1	1	0	0	0	0	0	1	0	0
$10 < M \leq 16$	$800 \leq VCO < 1000$	1	1	1	0	0	0	0	0	0	1
$10 < M \leq 16$	$1000 \leq VCO < 1200$	1	1	1	0	0	0	0	0	1	0
$10 < M \leq 16$	$1200 \leq VCO < 1500$	1	1	0	1	0	0	0	0	0	0
$10 < M \leq 16$	$1500 \leq VCO < 1800$	1	1	0	1	0	0	0	0	0	0

Table 10-5. CPR0_PLLC[TUNE] Bit Settings (Continued)

M Value Range	VCO Frequency Range (MHz)	TUNE									
		9	8	7	6	5	4	3	2	1	0
16 < M ≤ 32	600 ≤ VCO < 700	1	1	0	0	0	0	0	1	0	0
16 < M ≤ 32	700 ≤ VCO < 800	1	1	0	0	0	1	0	0	0	0
16 < M ≤ 32	800 ≤ VCO < 1000	1	1	1	0	0	0	0	0	1	1
16 < M ≤ 32	1000 ≤ VCO < 1200	1	1	1	0	0	1	0	0	0	0
16 < M ≤ 32	1200 ≤ VCO < 1500	1	1	0	1	0	0	0	0	0	0
16 < M ≤ 32	1500 ≤ VCO < 2000	1	1	0	1	0	0	0	0	0	1

10.2.5 IIC Bootstrap Controller Clocking

SysClk is divided by six to provide a clock for the IIC bootstrap controller. This clock is internal to the PPC460EX/EXr/GT and cannot be adjusted. Also, the IIC bootstrap controller further divides its clock by 128. Thus, for SysClk frequencies below 76.8MHz, a 100kHz serial ROM can be used. For SysClk frequencies above 76.8MHz, a 400 kHz serial ROM must be used.

10.2.6 Source Clocks for UART, Timers, MDIO, IIC and SPI

The UART serial clocks can be generated either from an internal clock divider circuit or from the UARTSerClk chip input. See *Serial Port Clocking* on page 1090 for instructions on selecting the UART serial clock source and configuring the UART serial clock divisors.

There are two timers, the Timer Facilities of the PPC440 Processor and the General Purpose Timer (GPT). The Timer Facility derives its time base from either the CPUClk or the input TmrClk. The clock source is selected by CCR1[TCS] as described in the *PPC440H6 Processor User's Manual*. The GPT time base is derived from the OPBClk.

The Ethernet management interface clock (GMCMDClk) is derived from the OPBClk. The EMACxMR1[OBCI] bit field contains the OPB to GMCMDClk divisor.

After reset, the IIC output clocks (IICxSCLK) are derived from the OPBClk. The IICx_CLKDIV register contains the OPB to IICxSCLK divisor.

The SPI output clock (SPIClkOut) is derived from the OPBClk. The SPI0_CDM register contains the OPB to SPIClkOut divisor.

10.2.7 Clocks For DDR SDRAM Controller

The DDR memory controller generates a differential MemClkOut to the DDR SDRAM chips at the DDR1xClk frequency. For 1:1 DDR:PLB ratio, set CPR0_PLB2D[PLB2DV0] = divide by 1 and SDR0_DDR0[DDRM] = 1:1 ratio. SDR0_DDR0[DDRM] by default is set to 1:1 DDR:PLB ratio.

10.2.8 SYS_PLL Strapping

System clocking is controlled primarily by the CPR0_PLLC, CPR0_PLLD, CPR0_AHBD, CPR0_OPBD, CPR0_PERD, and SDR0_CP440 registers. These registers are normally initialized at power on using the IIC bootstrap controller and an external serial ROM. The SDR0_SDSTP0 and SDR0_SDSTP1 registers are also initialized during this process. If a serial ROM is not present at power on, then the pre-programmed defaults will be used.

System reset always reloads SDR0_SDSTP0, SDR0_SDSTP1, SDR0_CUST0 and SDR0_CUST1 using the values programmed into the serial bootstrap ROM.

Several bit values in the clocking registers are required to correctly configure the chip in a way that enables reliable operation of SYS_PLL and clock divider circuitry. Setting these values incorrectly may render the chip unusable. See the following section for assistance in selecting acceptable system clocking divider ratios.

Note: The clock configuration can be changed through software using the steps outlined in *Bootstrap Options* on page 215 and *Clocking Update Register (CPR0_CLKUPD)* on page 247.

10.2.9 PLL Bypass (Emulation Mode)

The CPR0_PLLD[ENG] bit is provided for disabling the SYS_PLL if it is intentionally being used outside of its operating range. The most common reason for using this mode is for emulation systems used in product development, in which the SysClk input to chip is set to a frequency well below 66.67 MHz (perhaps as low as 1 MHz). Forcing the SYS_PLL into bypass mode directly feeds SysClk to the divider logic, allowing parts of the chip to operate well outside of its supported frequency range.

10.2.10 System Clock Ratios

The clocking logic permits CPU:PLB clock frequencies in the ratio of N:1, where N = 2, 3, 4, 5, 6, 7, or 8. These ratios are set by the values chosen for the divisors in the clocking tree. The clocking logic also supports clock bus ratios on PLB:AHB, PLB:OPB, and OPB:PER. The following table indicates the system clock ratios supported in PPC460EX/EXr/GT.

Table 10-6. System Clock Ratios

CPU:PLB Ratio	DDR:PLB Ratio	PLB:AHB Ratio	PLB:OPB Ratio	OPB:PER Ratio
N:1 N = 2, 3, 4, 5, 6, 7, or 8	1:1	N:1 N = 1 or 2	N:1 N = 1, 2, 3, or 4	N:1 N = 1, 2, 3, or 4

10.2.11 Choosing System Clock Ratios

Table 10-7 describes the equations that can be used to determine the resultant VCO, CPU, PLB frequencies based on selected divisor settings.

Table 10-7. Equations to Determine VCO, CPU, PLB Frequency

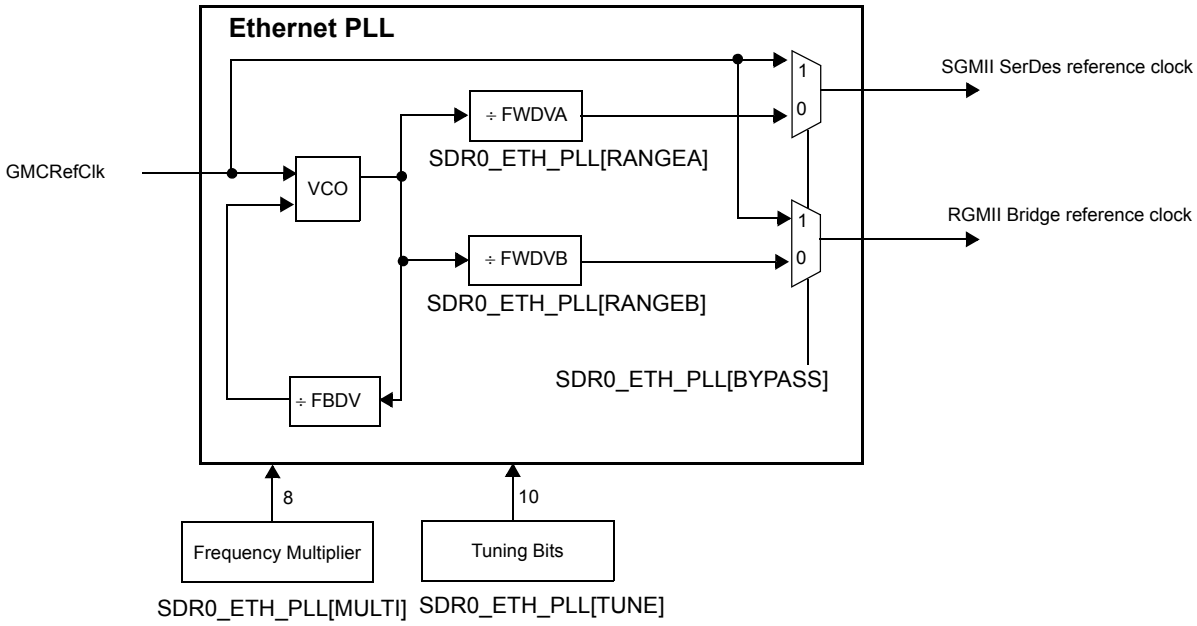
Feedback Selection	Equations
PLL Local	M = FBDV
PerClk	M = FWDVA x PLBEDV0 x OPBDV0 x PERDV0 (FBDV = 1)
For both: VCO = SysClk x M CPU = VCO/FWDVA Configure such that PLBClk = PLBClk_A = PLBClk_B. PLBClk_B = VCO/FWDVB/PLB2DV0/2 PLBClk_A = VCO/FWDVA/PLBEDV0 PLBEDV0 = (FWDVB/FWDVA) x PLB2DV0 x 2 CPU: PLB = N:1, N = PLBEDV0 For DDR:PLB = 1:1, set PLB2DV0 = ÷1 and SDR0_DDR0[DDRM] = 1:1 ratio.	
Note: PerClk feedback is not recommended.	

10.3 Ethernet Clocking

The Ethernet PLL generates two fixed frequencies for the Ethernet Subsystem: 1) 1250MHz for SGMII, and 2) 125MHz for GMII and RGMII modes. Depending on the mode, these clocks my be divided down further for use by the logic. The Ethernet reference clock (GMCRefClk) is required to be 125MHz. The VCO operates at 1250MHz in order to produce the two clock outputs for the SGMII SerDes and RGMII Bridge.

During reset, the output of the PLL is replaced with a slower version of the system clock, SYSClk/4, which is used to initialize the Ethernet Subsystem while the PLL clocks are stabilizing. After reset, the Ethernet PLL is configured by default. No initialization of SDR0_ETH_PLL is required.

Figure 10-2. Ethernet Clocking



Note: No initialization of SDR0_ETH_PLL required. The SDR0_ETH_PLL is configured by default after reset.

10.4 System (CPU) Clocking Registers

Table 10-8 lists the clocking configuration address and data registers, which are used to access the non-PCI Express clocking registers.

The change the clock configuration through software see *Bootstrap Options* on page 215 and *Clocking Update Register (CPR0_CLKUPD)* on page 247.

Table 10-8. Clocking Control Register Access

Mnemonic	Register	DCR Address	Access	Page
CPR0_CFGADDR	Clocking Configuration Address Register	0x000C	R/W	246
CPR0_CFGDATA	Clocking Configuration Data Register	0x000D	R/W	246

Table 10-9 lists the indirectly accessed DCRs, which control clocking and power-on reset.

The following code illustrates how to access a CPR0 register by writing the CPR0_CFGDATA register and then reading back the written value:

```
li r3, CPR0_DCR_Offset          ! address offset of CPR0 register
lis r4,<config upper>           ! upper half of configuration data
ori r4, r4,<config lower>       ! lower half of configuration data
mtdcr CPR0_CFGADDR, r3          ! set offset address
mtdcr CPR0_CFGDATA, r4          ! write configuration data
mfdcr r5, CPR0_CFGDATA          ! read back configuration data
```

Table 10-9. Clocking Control Registers

Mnemonic	Register	DCR Offset	Access	Page
CPR0_CLKUPD	Clocking Update Register	0x0020	R/W	247
CPR0_PLLC	SYS_PLL Control Register	0x0040	R/W	247
CPR0_PLLD	SYS_PLL Divider Register	0x0060	R/W	248
CPR0_PLBED	PLB Early Clock Divider Register	0x0080	R/W	249
CPR0_PLB2D	PLB Clock Divider Register	0x00A0	R/W	250
CPR0_OPBD	OPB Clock Divider Register	0x00C0	R/W	250
CPR0_PERD	Peripheral Clock Divider Register	0x00E0	R/W	250
CPR0_AHBD	AHB Clock Divider Register	0x0100	R/W	250
CPR0_ICFG	Initial Configuration Register	0x0140	R/W	251

10.4.1 Clock/Power-On Reset Configuration Address Register (CPR0_CFGADDR)

This is a 32-bit register used to access the CPR0_xxxx control registers.

Figure 10-3. Clock/Power-On Reset Configuration Address Register (CPR0_CFGADDR)

0:16		Reserved	
17:31	OFST	Offset	

10.4.2 Clock/Power-On Reset Configuration Data Register (CPR0_CFGDATA)

This is a 32-bit register is used to access the CPR0_xxxx control registers.

Figure 10-4. Clock/Power-On Reset Configuration Data Register (CPR0_CFGDATA)

0:31		Data	

10.4.3 Clocking Update Register (CPR0_CLKUPD)

CPR0_CLKUPD enables software to configure the clock settings in CPR0_AHBD, CPR0_OPBD, and CPR0_PERD without performing a reset. Software that modifies CPR0_OPBD or CPR0_PERD registers must not execute from a device attached to the peripheral bus such as FLASH when writing these registers. The software to perform this function should be cached in On Chip Memory or should execute from DDR memory.

Note: When using PerClk feedback (CPR0_PLLC[SEL] = 0b11), CPR0_PPBD and CPR0_PERD must not be changed.

Figure 10-5. Clocking Update Register (CPR0_CLKUPD)

0	BSY	Clocking Subsystem Busy 0 Stable 1 Making changes	In Read Access mode. If CPR0_CLKUPD[BSY] = 1, software needs to wait until CPR0_CLKUPD[BSY] = 0 before initiating any additional changes.
	CUD	Dividers Clocking Update Delay 0 Disabled 1 Enabled	In Write Access mode. If CPR0_CLKUPD[CUD] = 1, clocking update action is taken and the clocking logic is instructed to begin changing clocks to the newly programmed divider values for any dividers immediately outside the PLL. **Result:** System clocks are updated at the next coincident rising edge on all output clocks using the currently programmed values in the CPR0 divider registers.
1	CUI	PLL Setting Clocking Update Immediate 0 Disabled	This feature is not supported and the bit is always set to and read as 0.
2:31		Reserved	

Note: The CPR0_CLKUPD register is unique, in that writes automatically result in an update to the clocking subsystem, while reads of the same bit(s) report a status.

10.4.4 SYS_PLL Control Register (CPR0_PLLC)

Figure 10-6. SYS_PLL Control Register (CPR0_PLLC)

0	RST	Reset 0 PLL allowed to lock 1 PLL is forced into reset.	For the CPR clocking logic, the reset value of the RST bit is the complement of the ENG bit.
1	ENG	Engage 0 SysClk is the source for primary forward divisor 1 PLL's VCO is the source for primary forward divisor	
2:5		Reserved	
6:7	SEL	Feedback Selection 00 PLL local feedback 01 Reserved 10 Reserved 11 PerClk feedback	Using PLL local feedback source implies that the PLL is not being used to adjust the generated clocks to be phase aligned with SysClk. **Note:** PerClk feedback is not recommended.
8:21		Reserved	
22:31	TUNE	TUNE bits	See *Table 10-5* on page 241 for tune bit settings.

10.4.5 SYS_PLL Divider Register (CPR0_PLLD)

Figure 10-7. SYS_PLL Divider Register (CPR0_PLLD)

0:7	FBDV	PLL Feedback Divisor X000_0000 1 1111_1111 2 0111_1110 3 1111_1101 4 0111_1010 5 1111_0101 6 0110_1010 7 1101_0101 8 0010_1010 9 1101_0100 10 0010_1001 11 1101_0011 12 0010_0110 13 1100_1100 14 0001_1001 15 1011_0011 16 0110_0111 17 1100_1110 18 0001_1101 19 1011_1011 20 0111_0111 21 1110_1110 22 0101_1101 23 1011_1010 24 0111_0100 25 1110_1001 26 0101_0010 27 1010_0101 28 0100_1011 29 1001_0110 30 0010_1100 31 1101_1000 32 .... 1001_1111 254 0011_1111 255	All practical clock configurations are possible within the range of 1 to 32.
8:11		Reserved	

12:15	FWDVA	PLL Forward Divisor A 0000 1 0001 2 1111 3 0100 4 1001 5 1010 6 1101 7 1110 8 0011 9 1100 10 0101 11 1000 12 0111 13 0010 14 1011 15 0110 16	
16:19		Reserved	
20:23	FWDVB	PLL Forward Divisor B 0000 1 0001 2 1111 3 0100 4 1001 5 1010 6 1101 7 1110 8 0011 9 1100 10 0101 11 1000 12 0111 13 0010 14 1011 15 0110 16	
24:31		Reserved	

10.4.6 PLB Early Clock Divider Register (CPR0_PLBED)

Figure 10-8. PLB Early Clock Divider Register (CPR0_PLBED)			
0:4		Reserved	
5:7	PLBEDV0	PLB Early Clock Divisor 0 000 8 001 1 010 2 011 3 100 4 101 5 110 6 111 7	Set PLBEDV0 so that PLBClk_A and PLBClk_B are the same frequency.
8:31		Reserved	

10.4.7 PLB2 Clock Divider Register (CPR0_PLB2D)

Figure 10-9. PLB2 Clock Divider Register (CPR0_PLB2D)

0:6		Reserved	
7	PLB2DV0	PLB Clock Divisor 0 0 2 1 1	PLB2DV0 must be set to 1.
8:31		Reserved	

10.4.8 OPB Clock Divider Register (CPR0_OPBD)

Figure 10-10. OPB Clock Divider Register (CPR0_OPBD)

0:5		Reserved	
6:7	OPBDV0	OPB Clock Divisor 0 00 4 01 1 10 2 11 3	The OPB Clock must be 66 MHz or greater to operate the Ethernet at 1 Gbit/s.
8:31		Reserved	

10.4.9 Peripheral Clock Divider Register (CPR0_PERD)

Figure 10-11. Peripheral Clock Divider Register (CPR0_PERD)

0:5		Reserved	
6:7	PERDV0	Peripheral Clock Divisor 0 00 4 01 1 10 2 11 3	
8:31		Reserved	

10.4.10 AHB Clock Divider Register (CPR0_AHBD)

Figure 10-12. AHB Clock Divider Register (CPR0_AHBD)

0:6		Reserved	
7	AHBDV0	AHB Clock Divisor 0 0 2 1 1	
8:31		Reserved	

10.4.11 Initial Configuration Register (CPR0_ICFG)

Figure 10-13. Initial Configuration Register (CPR0_ICFG)

0	RLI	Reload Inhibit 0 Reset CPR registers from configuration source defined by CPR0_ICFG[ICS] 1 Ignore CPR0_ICFG[ICS] configuration source and preserve current values in CPR0 registers	CPR0_ICFG[RLI] preserves contents of CPR0 registers during a chip reset. See section *Bootstrap Options* on page 215 for instructions on how to configure the CPR0 registers.
1:28		Reserved	
29	ICS1	Initial value of the strap pin $\overline{\text{UART0CTS}}$	Read only
30	ICS2	Initial value of the strap pin $\overline{\text{UART0DCD}}$	Read only
31	ICS3	Initial value of the strap pin $\overline{\text{UART0DSR}}$	Read only

Part III. System Operations

11. Interrupt Controller Operations

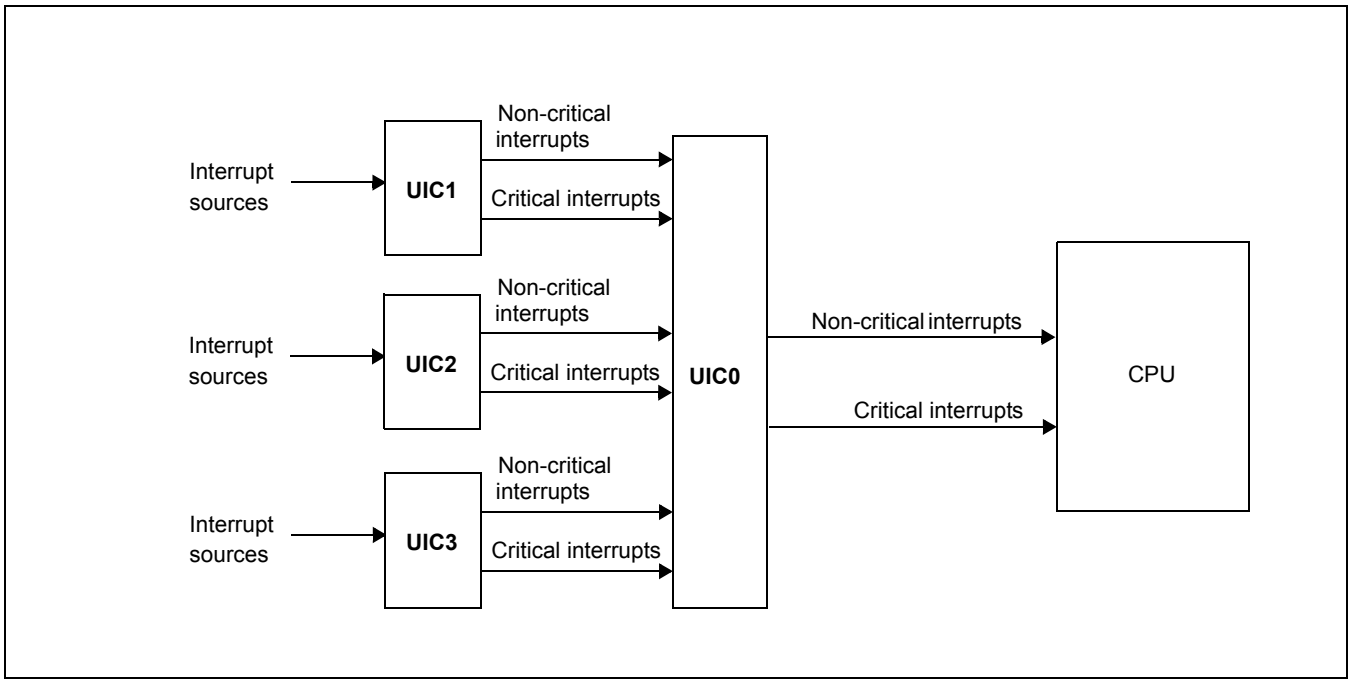
The PPC460EX/EXr/GT contains four universal interrupt controllers (UIC0, UIC1, UIC2, and UIC3) that provide all necessary control, status, and communication between the various internal and external interrupt sources and the processor.

11.1 UIC Overview

The UICs support multiple internal and external interrupts. Status reporting (using the UIC Status Register [UICn_SR]) is provided to ensure that systems software can determine the current and interrupting state of the system and respond appropriately. Software can generate interrupts to simplify software development and for diagnostics.

UIC0 collects interrupts from internal and external sources, including the critical and non-critical interrupt outputs of the secondary interrupt controllers. The UICs are cascaded as shown in *Figure 11-1* below:

Figure 11-1. Cascaded UIC Organization



The interrupts can be programmed, using the UIC Critical Register (UICn_CR), to generate either a critical or a non-critical interrupt signal to the processor.

The privileged **mtdcr** and **mfdcr** instructions, which are used by system software, are used to read and write the UIC registers.

An optional critical interrupt vector generator can reduce interrupt handling latency for critical interrupts. Vector calculation is described in detail in *UICn Vector Register (UICn_VR)* on page 265.

11.2 UIC Features

- Support for 16 external interrupts
- Support for asynchronous level- or edge-sensitive interrupt types
- Programmable polarity for all interrupt types
- Programmable critical/non-critical interrupt selection for each interrupt bit
- Prioritized critical interrupt vector generation
- A UIC Status Register (UICn_SR) providing the following information:
  - Current state of interrupts
  - Current state of all enabled interrupts (those masked using the UIC Enable Register (UICn_ER))

11.3 UIC Interrupt Assignments

The UIC supports internal and external interrupt sources as shown in the following tables.

Table 11-1. UIC0 Interrupt Assignments

Interrupt	Polarity	Sensitivity	Interrupt Source
0	N/A	N/A	unused
1	High	Level	UART1
2	High	Level	IIC 0
3	High	Level	IIC 1
4	High	Level	PCI0 Inbound Message
5	High	Level	PCI0 Command Write Register
6	High	Level	PCI0 Power Management
7	Rising	Edge	PCI0 VPD Access
8	Rising	Edge	PCI0 MSI Level 0
9	Programmable	Programmable	External IRQ 0
10	High	Level	UIC2 Non-Critical interrupt
11	High	Level	UIC2 Critical Interrupt
12	High	Level	DMA2P40 Channel 0
13	High	Level	DMA2P40 Channel 1
14	High	Level	DMA2P40 Channel 2
15	High	Level	DMA2P40 Channel 3
16	High	Level	UIC3 Non-Critical interrupt
17	High	Level	UIC3 Critical Interrupt
18	Programmable	Programmable	External IRQ 1
19	High	Level	TRNG Data Available
20	Rising	Edge	PKA Ready (PKA[1])
21	High	Level	HSDMA Command Pointer FIFO Full
22	High	Level	HSDMA Command Status FIFO Needs Service
23	High	Level	I2O Inbound Doorbell
24	High	Level	I2O Inbound Post List FIFO Not Empty
25	High	Level	I2O Region 0 Low Latency PLB Write
26	High	Level	I2O Region 1 Low Latency PLB Write
27	High	Level	I2O Region 0 High Bandwidth PLB Write
28	High	Level	I2O Region 1 High Bandwidth PLB Write
29	High	Level	Security EIP-94
30	High	Level	UIC1 Non-Critical interrupt
31	High	Level	UIC1 Critical Interrupt

Table 11-2. UIC1 Interrupt Assignments

Interrupt	Polarity	Sensitivity	Interrupt Source
0	Programmable	Programmable	External IRQ 2
1	High	Level	UART0
2	High	Level	SPI
3	High	Level	TRNG Alarm
4	High	Level	Correctable or Uncorrectable ECC Memory Error
5	High	Level	External Bus Controller (EBC)
6	High	Level	NDFC
7	High	Level	EIPPKP Slave Error
8	Rising	Edge	PCI0 MSI Level 1
9	Rising	Edge	PCI0 MSI Level 2
10	Rising	Edge	PCI0 MSI Level 3
11	Rising	Edge	L2 Cache
12	Rising	Edge	GPT Compare Timer 0
13	Rising	Edge	GPT Compare Timer 1
14	Rising	Edge	GPT Compare Timer 2
15	Rising	Edge	GPT Compare Timer 3
16	Rising	Edge	GPT Compare Timer 4
17	Rising	Edge	GPT Compare Timer 5
18	Rising	Edge	GPT Compare Timer 6
19	Rising	Edge	GPT Down Count Timer
20	Programmable	Programmable	External IRQ 3
21	Programmable	Programmable	External IRQ 4
22	High	Level	HSDMA Error
23	High	Level	I2O Error
24	High	Level	Serial ROM Error
25	High	Level	PCI0 Asynchronous Error (PCI0 ERROR_INT)
26	Programmable	Programmable	External IRQ 5
27	Programmable	Programmable	External IRQ 6
28	High	Level	UART2
29	High	Level	UART3
30	Programmable	Programmable	External IRQ 7
31	Programmable	Programmable	External IRQ 8

Table 11-3. UIC2 Interrupt Assignments

Interrupt	Polarity	Sensitivity	Interrupt Source
0	High	Level	TAHOE0
1	High	Level	TAHOE1
2	Programmable	Programmable	External IRQ 9
3	High	Level	MAL SERR
4	High	Level	MAL TXDE
5	High	Level	MAL RXDE
6	High	Level	MAL TX EOB
7	High	Level	MAL RX EOB
8	Rising	Edge	MAL Interrupt Coalescence TX0
9	Rising	Edge	MAL Interrupt Coalescence TX1
10	Rising	Edge	MAL Interrupt Coalescence TX2
11	Rising	Edge	MAL Interrupt Coalescence TX3
12	Rising	Edge	MAL Interrupt Coalescence RX0
13	Rising	Edge	MAL Interrupt Coalescence RX1
14	Rising	Edge	MAL Interrupt Coalescence RX2
15	Rising	Edge	MAL Interrupt Coalescence RX3
16	High	Level	EMAC0
17	High	Level	EMAC1
18	High	Level	EMAC2
19	High	Level	EMAC3
20	High	Level	EMAC0 Wake-Up
21	High	Level	EMAC1 Wake-Up
22	High	Level	EMAC2 Wake-Up
23	High	Level	EMAC3 Wake-Up
24	Programmable	Programmable	External IRQ 10
25	Programmable	Programmable	External IRQ 11
26			Reserved
27	High	Level	PLB4XAHB/AHBARB Error
28	High	Level	USB2.0 OTG
29	High	Level	USB2.0 Host EHCI
30	High	Level	USB2.0 Host OHCI
31	High	Level	USB2.0 Host OHCI SMI

Table 11-4. UIC3 Interrupt Assignments

Interrupt	Polarity	Sensitivity	Interrupt Source
0	High	Level	PE0 AL/SATA (PPC460EX only)
1	Rising	Edge	PE0 VPD Access
2	Rising	Edge	PE0 Hot Reset Request
3	High	Level	PE0 TCR
4	Falling	Edge	PE0 BusMaster VCO
5	High	Level	PE0 DCR Error/AHBDMAC (PPC460EX SATA only)
6	High	Level	PE1 AL/SRIO Bridge (PPC460GT only)
7	Rising	Edge	PE1 VPD Access
8	Rising	Edge	PE1 Hot Reset Request
9	High	Level	PE1 TCR/SRIO Logic Layer (PPC460GT only)
10	Falling	Edge	PE1 BusMaster VCO
11	High	Level	PE1 DCR Error/SRIO DCR Error (PPC460GT only)
12	High	Level	PE0 INTA
13	High	Level	PE0 INTB
14	High	Level	PE0 INTC
15	High	Level	PE0 INTD
16	High	Level	PE1 INTA
17	High	Level	PE1 INTB
18	High	Level	PE1 INTC
19	High	Level	PE1 INTD
20	Programmable	Programmable	External IRQ 12
21	Programmable	Programmable	External IRQ 13
22	Programmable	Programmable	External IRQ 14
23	Programmable	Programmable	External IRQ 15
24	Rising	Edge	PCI Express MSI Level 0
25	Rising	Edge	PCI Express MSI Level 1
26	Rising	Edge	PCI Express MSI Level 2
27	Rising	Edge	PCI Express MSI Level 3
28	Rising	Edge	PCI Express MSI Level 4
29	Rising	Edge	PCI Express MSI Level 5
30	Rising	Edge	PCI Express MSI Level 6
31	Rising	Edge	PCI Express MSI Level 7

11.4 Interrupt Programmability

All of the on-chip interrupts and the external IRQs are programmable. However, the polarity and sensitivity of the on-chip interrupts must be programmed as shown in *Table 11-1*, *Table 11-2*, *Table 11-3*, and *Table 11-4* using the UIC Polarity Register (UICn_PR) and UIC Trigger Register (UICn_TR).

11.5 UIC Registers

The UIC is controlled through the Device Control Registers (DCRs) listed in *Table 11-5*. The registers are accessed using the **mfdcr** and **mtdcr** instructions.

Table 11-5. UIC Device Control Registers

Mnemonic	Register	Address	Access	Page
UIC0_SR	UIC Status Register 0	0x0C0	Read/Clear	262
UIC1_SR	UIC Status Register 1	0x0D0	Read/Clear	262
UIC2_SR	UIC Status Register 2	0x0E0	Read/Clear	262
UIC3_SR	UIC Status Register 3	0x0F0	Read/Clear	262
UIC0_SSR	UIC Status Set Register 0	0x0C1	Read/Set	262
UIC1_SSR	UIC Status Set Register 1	0x0D1	Read/Set	262
UIC2_SSR	UIC Status Set Register 2	0x0E1	Read/Set	262
UIC3_SSR	UIC Status Set Register 3	0x0F1	Read/Set	262
UIC0_ER	UIC Enable Register 0	0x0C2	R/W	262
UIC1_ER	UIC Enable Register 1	0x0D2	R/W	262
UIC2_ER	UIC Enable Register 2	0x0E2	R/W	262
UIC3_ER	UIC Enable Register 3	0x0F2	R/W	262
UIC0_CR	UIC Critical Register 0	0x0C3	R/W	263
UIC1_CR	UIC Critical Register 1	0x0D3	R/W	263
UIC2_CR	UIC Critical Register 2	0x0E3	R/W	263
UIC3_CR	UIC Critical Register 3	0x0F3	R/W	263
UIC0_PR	UIC Polarity Register 0	0x0C4	R/W	263
UIC1_PR	UIC Polarity Register 1	0x0D4	R/W	263
UIC2_PR	UIC Polarity Register 2	0x0E4	R/W	263
UIC3_PR	UIC Polarity Register 3	0x0F4	R/W	263
UIC0_TR	UIC Trigger Register 0	0x0C5	R/W	264
UIC1_TR	UIC Trigger Register 1	0x0D5	R/W	264
UIC2_TR	UIC Trigger Register 2	0x0E5	R/W	264
UIC3_TR	UIC Trigger Register 3	0x0F5	R/W	264
UIC0_MSR	UIC Masked Status Register 0	0x0C6	Read-only	264
UIC1_MSR	UIC Masked Status Register 1	0x0D6	Read-only	264
UIC2_MSR	UIC Masked Status Register 2	0x0E6	Read-only	264
UIC3_MSR	UIC Masked Status Register 3	0x0F6	Read-only	264

Table 11-5. UIC Device Control Registers (Continued)

Mnemonic	Register	Address	Access	Page
UIC0_VR	UIC Vector Register 0	0x0C7	Read-only	265
UIC1_VR	UIC Vector Register 1	0x0D7	Read-only	265
UIC2_VR	UIC Vector Register 2	0x0E7	Read-only	265
UIC3_VR	UIC Vector Register 3	0x0F7	Read-only	265
UIC0_VCR	UIC Vector Configuration Register 0	0x0C8	Write-only	264
UIC1_VCR	UIC Vector Configuration Register 1	0x0D8	Write-only	264
UIC2_VCR	UIC Vector Configuration Register 2	0x0E8	Write-only	264
UIC3_VCR	UIC Vector Configuration Register 3	0x0F8	Write-only	264

The following registers descriptions contain bit settings corresponding to the UIC assignments as defined in *Table 11-1*, *Table 11-2*, *Table 11-3*, and *Table 11-4*.

11.5.1 UICn Status Register (UICn_SR)

To report interrupt status, the UICn_SR fields capture and hold internal and external interrupts until the fields are intentionally reset. To reset a field, write 1 to the field.

The values of other UIC registers do not affect UICn_SR fields.

Figure 11-2. UICn Status Register (UICn_SR)

0:31	Label	Interrupt status 0 Interrupt has not occurred 1 Interrupt occurred	Each status bit corresponds to the interrupt assigned to UICn where n = 0, 1, 2, or 3.

11.5.2 UICn Set Status Register (UICn_SSR)

For the purposes of debug, a means of setting the UICn_SR fields is necessary. To set a field, write 1 to the field. Writing a 0 to a field has no effect on the field. Reading UICn_SSR returns the contents of the UICn_SR register.

Figure 11-3. UICn Set Status Register (UICn_SSR)

0:31	Label	Interrupt set status 0 Interrupt is not affected 1 Interrupt is set	

11.5.3 UICn Enable Register (UICn_ER)

The fields of UICn_ER, which correspond to the fields of the UICn_SR, enable or disable the reporting of the corresponding fields of the UICn_SR.

If a UICn_ER field is set to 1, the corresponding field of UICn_SR generates a critical or non-critical interrupt signal to the processor, if UICn_SR field is set to 1. If a UICn_ER field is set to 0, the corresponding field of UICn_SR does not generate a critical or non-critical interrupt signal to the processor, regardless of the setting of UICn_SR field. The critical and non-critical interrupt signals in the processor are controlled by fields in the Machine State Register (MSR).

The class of generated signals (critical or non-critical) is controlled by UICn_CR.

Figure 11-4. UICn Enable Register (UICn_ER)			
0:31	Label	Interrupt enable 0 Interrupt is disabled 1 Interrupt is enabled	Each enable bit corresponds to the interrupt assigned to UICn where n = 0, 1, 2, or 3.

11.5.4 UICn Critical Register (UICn_CR)

The fields of UICn_CR, which correspond to the fields of UICn_SR and UICn_ER, determine whether an interrupt captured in the corresponding fields of UICn_SR generates a non-critical or critical interrupt, if the interrupts are enabled in the corresponding fields of UICn_ER. The processor handles non-critical interrupts when MSR[EE] = 1 and critical interrupts when MSR[CE] = 1.

If a UICn_CR field is set to 0, an enabled interrupt (captured in the corresponding field of UICn_SR and enabled in the corresponding field of UICn_ER) generates a non-critical interrupt signal to the processor. If a UICn_CR field is a 1, a critical interrupt signal is generated.

Figure 11-5. UICn Critical Register (UICn_CR)			
0:31	Label	Interrupt class 0 Interrupt is non-critical 1 Interrupt is critical	Each class bit corresponds to the interrupt assigned to UICn where n = 0, 1, 2, or 3.

11.5.5 UICn Polarity Register (UICn_PR)

The fields of UICn_PR, which correspond to the fields of UICn_SR, determine whether the corresponding fields in UICn_SR have a positive or negative polarity.

For level-sensitive interrupts, a 0 in a UICn_PR field causes the corresponding interrupt to be negative active. A 1 in a UICn_PR field causes the corresponding interrupt to be positive active.

For edge-sensitive interrupts, a 0 in a UICn_PR field causes the corresponding interrupt to be detected on a falling edge (as polarity changes from 1 to 0). A 1 in a UICn_PR field causes the corresponding interrupt to be detected on a rising edge (as polarity changes from 0 to 1).

Because the on-chip interrupts (those controlled by UICn_PR) have positive polarity, the associated fields must be set to 1.

Figure 11-6. UICn Polarity Register (UICn_PR)			
0:31	Label	Interrupt Polarity 0 Interrupt has negative polarity 1 Interrupt has positive polarity	Each polarity bit corresponds to the interrupt assigned to UICn where n = 0, 1, 2, or 3.

11.5.6 UICn Trigger Register (UICn_TR)

The fields of UICn_TR, which correspond to the fields of UICn_SR, determine whether corresponding fields in UICn_SR are edge-sensitive or level-sensitive.

Edge-sensitive interrupts are triggered depending on whether the associated interrupt signal is rising or falling (changing from 0 to 1 or 1 to 0, respectively). Whether a rising or falling edge causes the trigger is controlled by bits in UICn_PR.

Level-sensitive interrupts are triggered depending on whether the associated interrupt signal is high (1) or low (0).

If a UICn_TR field is 0, the associated interrupt is level-sensitive. If a UICn_TR field is 1, the interrupt is edge-sensitive.

Figure 11-7. UICn Trigger Register (UICn_TR)

0:31	Label	Interrupt Trigger 0 Interrupt is level sensitive 1 Interrupt is edge sensitive	Each trigger bit corresponds to the interrupt assigned to UICn where n = 0, 1, 2, or 3.

11.5.7 UICn Masked Status Register (UICn_MSR)

This read-only register contains the result of masking UICn_SR with UICn_ER. Reading this register, instead of the actual UICn_SR, eliminates the need for software to read and apply the enable mask to the contents of UICn_SR to determine which enabled interrupt fields are active.

If an interrupt is configured as level-sensitive, and a clear is attempted on UICn_SR, a UICn_SR field is not cleared if the incoming interrupt signal is at the asserted polarity. The interrupt signal must be reset before UICn_SR can be successfully cleared.

Figure 11-8. UICn Masked Status Register (UICn_MSR)

0:31	Label	Masked Interrupt Status 0 Masked interrupt has not occurred. 1 Masked interrupt occurred.	Each masked interrupt status bit corresponds to the interrupt assigned to UICn where n = 0, 1, 2, or 3.

11.5.8 UIC0 Vector Configuration Register (UICn_VCR)

The write-only UICn_VCR enables software control of interrupt vector generation for critical interrupts. UICn_VCR contains an address, used as an interrupt vector base address, and specifies interrupt ordering priority. Vector generation is not performed for non-critical interrupts.

UICn_VCR[VBA] can contain either the base address for an interrupt handler vector table or the base address for the interrupt handler associated with each interrupt. The actual interrupt vector (the address of the interrupt handler that services the interrupt) is generated in UICn_VR, using UICn_VCR[VBA]. Vector generation is described in *UICn Vector Register (UICn_VR)* on page 265. Because the two lowest-order bits of an interrupt handler address are assumed to be 00 to ensure word alignment, 30 bits are sufficient to form the base address.

A general interrupt handler uses the vector to access a table of interrupt vectors. Each interrupt vector table entry contains the address of an interrupt handler for a specific interrupt. Alternatively, UICn_VCR[VBA] can directly address the interrupt handlers for specific interrupts, which in memory are separated by an offset calculated in UICn_VR.

UICn_VCR[PRO] controls whether the interrupt associated with UICn_SR[0] or UICn_SR[31] has the highest priority. If UICn_VCR[PRO] = 0, the interrupt associated with UICn_SR[31] has the highest priority; if UICn_VCR[PRO] = 1, the interrupt associated with UICn_SR[0] has the highest priority. The bit closest to the highest priority field that is programmed in UICn_CR as a interrupt has the second highest priority. Priority decreases across UICn_SR to the end opposite the highest priority field.

Figure 11-9. UICn Vector Configuration Register (UICn_VCR)

0:29	VBA	Vector Base Address	
30		Reserved	
31	PRO	Priority Ordering 0 UICn_SR[31] is the highest priority interrupt. 1 UICn_SR[0] is the highest priority interrupt.	Vector generation is not performed for non-critical interrupts.

11.5.9 UICn Vector Register (UICn_VR)

The read-only UICn_VR contains an interrupt vector that can reduce interrupt handling latency for critical interrupts. Vector generation logic adds an offset to UICn_VCR[VBA], and the sum is returned in the UIC0_VR. Vectors are not computed for non-critical interrupts.

The interrupt vector is based on the field position of the current highest priority, enabled, active, critical interrupt relative to the highest priority interrupt in UICn_SR. The generated vectors can be programmed to point directly to the interrupt handlers.

Programming Note: Regardless of the programming of UICn_VCR and UICn_VR registers, the processor always vectors to IVPR and IVOR0 when a critical interrupt occurs.

The interrupt vector offset is based on the bit position of the current highest priority, enabled, active, critical interrupt relative to the highest priority interrupt in UICn_SR. The offset has a fixed value of 512 per bit. The main critical interrupt handler can interpret the vector returned by UICn_VR as the address of the interrupt handler for that interrupt, assuming the routine is 512 bytes or smaller. Alternatively, the main critical interrupt handler can interpret the vector as a look-up table entry for the address of the interrupt handler for that interrupt.

Figure 11-10. UIC Vector Register (UICn_VR)

0:31	VR	Interrupt Vector	

The following example illustrates the generation of a UICn_VR vector for external interrupt request IRQ2.

For the example, assume that UICn_VCR[PRO] = 0, so that UICn_SR[EIR6S] ($UICn\_SR_{31}$)has the highest interrupt priority, and that UICn_SR[EIR2S] ($UICn\_SR_{27}$) is the current highest priority, enabled, active, critical interrupt. To generate the vector for the interrupt associated with UICn_SR[EIR2S], internal logic multiplies the difference between the highest priority interrupt bit and the active enabled priority interrupt bit by 512. The interrupt vector offset is therefore $(31 - 27) \times 512 = 4 \times 512$. This offset is added to the base address in UICn_VCR[VBA], and UICn_VR returns UICn_VCR[VBA] + (4×512).

11.5.9.1 Using the Value in UICn_VR as a Vector Address or Entry Table Lookup

If an interrupt handler is 512 bytes or smaller, system software can interpret the value returned in UICn_VR as an address. In this case, when the interrupt is received, UICn_VR is read and software simply jumps to the address represented by the UICn_VR value. Alternatively, the routine can be at a different address, and system software can treat the value of UICn_VR as a pointer, storing the interrupt handler address in UICn_VR during system initialization. In this case, when the interrupt is handled, software must read UICn_VR, read the entry at the UICn_VR value, and jump to the entry. Hardware has no knowledge of the method is used, which is determined by system software.

11.5.9.2 Vector Generation Scenarios

For the following sequence, assume that the interrupts are enabled and critical (vectors are not generated for disabled or non-critical interrupts). The sequence illustrates several scenarios for vector generation.

1. An intermediate priority interrupt goes active; its vector is stored in UICn_VR.
2. A low priority interrupt goes active; UICn_VR is unchanged.
3. Software reads the vector; UICn_VR is unchanged.
4. Software resets the intermediate priority interrupt; UICn_VR contains the vector for the low priority interrupt.
5. A high priority interrupt goes active; UICn_VR contains the vector for the high priority interrupt.
6. Software resets the high priority interrupt; UICn_VR contains the vector for the low priority interrupt.
7. Software resets the UICn_ER field for the low priority interrupt, disabling it; UICn_VR contains 0x00000000.
8. UICn_CR is reprogrammed to make the low priority interrupt non-critical and UICn_ER is reprogrammed to re-enable the low priority interrupt; UICn_VR continues to contain 0x00000000.

12. Interrupt Handling

Interrupt and exception processing for all chip functions is handled by the PPC440 processor. Refer to the *PPC440H6 Processor User's Manual* for details.

13. Floating Point Unit Interrupts and Exceptions

An *interrupt* is the action in which the processor saves its old context (Machine State Register (MSR) and next instruction address NIA)) and begins execution at a pre-determined interrupt-handler address, with a modified MSR. *Exceptions* are the events that may cause the processor to take an interrupt, if the corresponding interrupt type is enabled.

Exceptions may be generated by the execution of instructions, or by signals from devices external to the PPC440 processor, the internal timer facilities, debug events, or error conditions.

13.1 Floating-Point Exceptions

Book-E requires all synchronous (precise and imprecise) interrupts to be reported in program order, as required by the sequential execution model. The only exception to this rule is the case of multiple synchronous imprecise interrupts. Upon a synchronizing event, all previously executed instructions are required to report any synchronous imprecise interrupt-generating exceptions, and the interrupt is then generated with all of those exception types reported cumulatively, in both the Exception Syndrome Register (ESR), and any status registers associated with the particular exception type, such as the Floating-Point Status and Control Register (FPSCR).

For any single instruction attempting to cause multiple exceptions for which the corresponding synchronous interrupt types are enabled, this section defines the priority order by which the instruction will be permitted to cause a *single* enabled exception, thus generating a particular synchronous interrupt. This exception priority mechanism, along with the requirement that synchronous interrupts must be generated in program order, guarantees that only one of the synchronous interrupt types is considered at any given time. The exception priority mechanism also prevents certain debug exceptions from existing in combination with certain other synchronous interrupt-generating exceptions.

This section does not define the permitted setting of multiple exceptions for which the corresponding interrupt types are disabled. The generation of exceptions for which the corresponding interrupt types are disabled has no effect on the generation of other exceptions for which the corresponding interrupt types are enabled. Conversely, if a particular exception for which the corresponding interrupt type is enabled is shown in the following sections to be of a higher priority than another exception, it will prevent the setting of that other exception, regardless of whether the corresponding interrupt type of the other exception is enabled or disabled.

Except as noted, only one of the exception types listed for a given instruction type can be generated at any given time. The priority of the exception types are listed in subsequent sections ranging from highest to lowest, within each instruction type.

Note: Some exception types may be mutually exclusive of each other and could otherwise be considered the same priority. In such cases, the exceptions are listed in the order suggested by the sequential execution model.

Computational instructions may cause exceptions. Aside from instructions that write the FPSCR, none of the noncomputational instructions can cause a floating-point exception.

All exceptions are handled precisely. Because this can affect performance adversely, it is strongly recommended that exceptions should be disabled when possible. This prevents the PPC440 FPU instruction stream from waiting for the execution of long latency instructions, such as **fdiv**[**s**].

13.2 Exceptions List

Book-E defines the following floating-point exceptions:

Table 13-1. Invalid Operation Exception Categories

Category	FPSCR Field
SNaN	VXSNAN
Infinity – Infinity	VXISI
Infinity ÷ Infinity	VXIDI
Zero ÷ Zero	VXZDZ
Infinity × Zero	VXIMZ
Invalid Compare	VXVC
Software Request	VXSOFT
Invalid Square Root	VXSQRT
Invalid Integer Convert	VXCVI

- Invalid Operation exception (VX)
- Zero Divide exception (ZX)
- Overflow exception (OX)
- Underflow exception (UX)
- Inexact exception (XI)

These exceptions can occur during execution of computational instructions. In addition, an Invalid Operation exception occurs when a **mtsfs** or **mtsfsi** instruction sets FPSCR[VXSOFT] = 1.

Each floating-point exception, and each category of Invalid Operation exception, has an exception bit in the FPSCR. Each floating-point exception also has a corresponding enable bit in the FPSCR. The exception bit indicates the occurrence of the corresponding exception. If an exception occurs, the corresponding enable bit controls the result produced by the instruction and, with MSR[FE0, FE1] whether and how the Enabled exception type Program interrupt is taken. (See *Floating-Point Exceptions* on page 269 for more information.) In general, the enabling specified by an enable bit is to enable the invoking the interrupt, not to enable the exception to occur. The occurrence of an exception depends only on the instruction and its inputs, not on the setting of any enable bits. The only exceptions to this general rule are the occurrence of an Underflow exception, which may depend on the setting of the enable bit, and the occurrence of a Inexact exception, which may depend on the Overflow exception bit not being set.

A single instruction, other than **mtfsf** or **mtfsfi**, can set more than one exception bit only in the following cases:

- An Inexact exception may be set with an Overflow exception.
- An Inexact Exception may be set with an Underflow exception.
- An Invalid Operation exception (SNaN) is set with Invalid Operation exception ($\infty \times 0$) for *Multiply-Add* instructions for which the values being multiplied are infinity and 0, and the value being added is an SNaN.
- An Invalid Operation exception (SNaN) can be set with Invalid Operation exception (Invalid Compare) for *Compare Ordered* instructions.

- Invalid Operation exception (SNaN) can be set with Invalid Operation exception (Invalid Integer Convert) for *Convert To Integer* instructions.

When an exception occurs, instruction execution may be suppressed or a result may be delivered, depending on the exception.

Instruction execution is suppressed for the following kinds of exception, so that there is no possibility that one of the operands is lost:

- Enabled Invalid Operation
- Enabled Zero Divide

For the remaining exceptions, a result is generated and written to the target specified by the instruction causing the exception. The result may be a different value for the enabled and disabled conditions for some of these exceptions. The exceptions that deliver a result are:

- Disabled Invalid Operation
- Disabled Zero Divide
- Disabled Overflow
- Disabled Underflow
- Disabled Inexact
- Enabled Overflow
- Enabled Underflow
- Enabled Inexact

Subsequent sections define each of the floating-point exceptions and specify the action that is taken when they are detected.

IEEE 754 specifies the handling of exceptional conditions in terms of "traps" and "trap handlers." In Book-E, an FPSCR exception enable bit of 1 causes generation of the result value specified in the IEEE standard for the "trap enabled" case. The exception is expected to be detected by software, which revises the result. An FPSCR exception enable bit of 0 causes generation of the "default result" value specified for the "trap disabled" (or "no trap occurs" or "trap is not implemented") case. Software is not expected to detect the exception, and simply uses the default result. The result to be delivered in each case for each exception is described in subsequent sections.

The IEEE 754 default behavior when an exception occurs is to generate a default value and to not notify software. In Book-E, if the IEEE 754 default behavior is desired for all exceptions, all FPSCR exception enable bits should be set to 0 and Ignore Exceptions Mode should be used (see *Table 13-2* on page 272). In this case, an Enabled exception type Program interrupt is not taken, even if floating-point exceptions occur. Software can inspect the FPSCR exception bits, if necessary, to determine whether exceptions have occurred.

If software is to be notified that a given kind of exception has occurred, the corresponding FPSCR exception enable bit must be set to 1 and a mode other than Ignore Exceptions Mode must be used. In this case, the Enabled exception type Program interrupt is taken if an enabled floating-point exception occurs. An Enabled exception type Program interrupt is also taken if an **mtsfs** or **mtsfsi** instruction sets an exception bit and its corresponding enable bit both to 1; the **mtsfs** or **mtsfsi** instruction is considered to cause the enabled exception.

MSR[FE0, FE1] control whether and how Enabled exception type Program interrupt are taken when an enabled floating-point exception occurs. An Enabled exception type Program interrupt is never taken because of a disabled floating-point exception.

Table 13-2. MSR[FE0, FE1] Modes

MSR[FE0]	MSR[FE1]	Mode
0	0	**Ignore Exceptions Mode** Floating-point exceptions do not cause an Enabled exception type Program interrupt to be taken.
1	1	**Precise Mode** An Enabled exception type Program interrupt is taken precisely at the instruction that caused the enabled exception.

If either MSR[FE0] or MSR[FE1] is 1, Enabled exception type Program interrupts are treated as in Precise Mode.

In all cases, the question of whether a floating-point result is stored, and what value is stored, is governed by the FPSCR exception enable bits, as described in subsequent sections, and is not affected by the value of MSR[FE0, FE1].

In all cases in which an Enabled exception type Program interrupt is taken, all instructions before the instruction at which the Enabled exception type Program interrupt is taken have completed, and no instruction after the instruction at which the Enabled exception type Program interrupt is taken has begun execution. The instruction at which the Enabled exception type Program interrupt is taken has not been executed unless it is the excepting instruction, in which case it has been executed if the exception is not an Enabled Invalid Operation exception or Enabled Zero Divide exception.

A **sync** instruction, or any other execution-synchronizing instruction or event, such as **isync**, also has the effects described above.

In order to obtain the best performance across the widest range of implementations, the programmer should follow these guidelines.

- If the IEEE 754 default results are acceptable to the application, Ignore Exceptions Mode should be used, with all FPSCR exception enable bits set to 0.
- Ignore Exceptions Mode should not, in general, be used when any FPSCR exception enable bits are set to 1.
- Precise Mode may degrade performance in some implementations, perhaps substantially, and therefore should be used only for debugging and other specialized applications.

13.3 Floating-Point Interrupts

The following interrupts are taken under the control of the PPC440 processor, and are not enabled by or reported in FPSCR bits:

- Floating-Point Unavailable
- Floating-Point Assist

13.3.1 Floating-Point Unavailable Interrupt

A Floating-Point Unavailable interrupt occurs when no higher priority exception exists, an attempt is made to execute a floating-point instruction (including floating-point loads, stores, and moves), and MSR[FP] = 0.

When a Floating-Point Unavailable interrupt occurs, the processor suppresses the execution of the instruction causing the Floating-Point Unavailable interrupt.

13.4 Floating-Point Exception Behavior

The following sections describe the behavior that results from the floating-point exceptions. For each exception, the definition of the exception is given, followed by a description of the action caused by the exception.

In general, each exception can result in either of two types of action, depending on whether the exception is enabled by its associated exception enable bit in the FPSCR.

13.4.1 Invalid Operation Exception

An Invalid Operation exception occurs when an operand is invalid for the specified operation. The invalid operations are:

- Any floating-point operation on a signaling NaN (SNaN)
- For add or subtract operations, magnitude subtraction of infinities ($\infty - \infty$)
- Division of infinity by infinity ($\infty \div \infty$)
- Division of zero by zero ($0 \div 0$)
- Multiplication of infinity by zero ($\infty \times 0$)
- Ordered comparison involving a NaN (Invalid Compare)
- Square root or reciprocal square root of a negative and nonzero number (Invalid Square Root)
- Integer conversion involving a number too large in magnitude to be represented in the target format, or involving an infinity or a NaN (Invalid Integer Convert)

In addition, an Invalid Operation exception occurs if software explicitly requests this by executing an **mtfsf**, **mtfsfi**, or **mtfsb1** instruction that sets FPSCR[VXSOFT] = 1.

> **Programming Note:** The purpose of FPSCR[VXSOFT] is to enable software to cause an Invalid Operation exception for a condition that is not necessarily associated with the execution of a floating-point instruction. For example, it may be set by a program that computes a square root, if the source operand is negative.

13.4.1.1 Action

The action taken depends on the setting of FPSCR[VE].

When Invalid Operation exception is enabled (FPSCR[VE] = 1) and an Invalid Operation exception occurs or software explicitly requests the exception, the following actions are taken:

- One or two FPSCR Invalid Operation exception bits, listed in *Table 13-3*, are set.

Table 13-3. Invalid Operation Exceptions

FPSCR Bit	Category
VXSNAN	SNaN
VXISI	Infinity – Infinity
VXIDI	Infinity ÷ Infinity
VXZDZ	Zero ÷ Zero
VXIMZ	Infinity × Zero
VXVC	Invalid Compare
VXSOFT	Software Request
VXSQRT	Invalid Square Root
VXCVI	Invalid Integer Convert

- If the operation is an arithmetic, **frsp**, or convert to integer operation, the target FPR is unchanged.
  - FPSCR[FR, FI] ← 0
  - FPSCR[FPRF] ← unchanged
- If the operation is a compare:
  - FPSCR[FR, FI, C] ← unchanged
  - FPSCR[FPCC] ← unordered
- If software explicitly requests the exception:

  FPSCR[FR, FI, FPRF] are as set by the **mtfsf**, **mtfsfi**, or **mtfsb1** instruction.

When Invalid Operation exception is disabled (FPSCR[VE] = 0) and an Invalid Operation exception occurs, or software explicitly requests the exception, the following actions are taken:

- One or two FPSCR Invalid Operation exception bits, listed in *Table 13-3*, are set.
- If the operation is an arithmetic or *Floating Round to Single-Precision* operation, the target FPR is set to a Quiet NaN
  - FPSCR[FR, FI] ← 0
  - FPSCR[FPRF] ← the class of the result (Quiet NaN)
- If the operation is a convert to 32-bit integer operation, the target FPR is set as follows:
  - $FPR(FRT)_{0:31}$ ← undefined
  - $FPR(FRT)_{32:63}$ are set to the most positive 32-bit integer if the operand in FPR(FRB) is a positive number or +∞, and to the most negative 32-bit integer if the operand in FPR(FRB) is a negative number, -∞, or NaN.
  - FPSCR[FR, FI] ← 0
  - FPSCR[FPRF] ← undefined
- If the operation is a compare:
  - FPSCR[FR, FI, C] ← unchanged
  - FPSCR[FPCC] ← unordered

- If software explicitly requests the exception:

  FPSCR[FR, FI, FPRF] are as set by the **mtfsf**, **mtfsfi**, or **mtfsb1** instruction.

13.4.2 Zero Divide Exception

A Zero Divide exception occurs when an **fdiv**[**s**] instruction is executed with a zero divisor value and a finite nonzero dividend value. This exception also occurs when a *Reciprocal Estimate* instruction (**fres** or **frsqrte**) is executed with an operand value of zero.

13.4.2.1 Action

The action to be taken depends on the setting of FPSCR[ZE].

When Zero Divide exception is enabled (FPSCR[ZE] = 1) and Zero Divide occurs, the following actions are taken:

- The Zero Divide exception bit is set.

  $FPSCR_{ZX} \leftarrow 1$
- $FPR(FRT)_{0:31} \leftarrow$ unchanged
- FPSCR[FR, FI] $\leftarrow 0$
- FPSCR[FPRF] $\leftarrow$ unchanged

When Zero Divide exception is disabled (FPSCR[ZE] = 0) and zero divide occurs, the following actions are taken:

- The Zero Divide exception bit is set.

  $FPSCR_{ZX} \leftarrow 1$
- FPR(FRT) $\leftarrow \pm$Infinity (the sign is determined by the XOR of the signs of the operands)
- FPSCR[FR, FI] $\leftarrow 0$
- FPSCR[FPRF] $\leftarrow$ class and sign of the result ($\pm$ Infinity)

13.4.3 Overflow Exception

Overflow occurs when the magnitude of what would have been the rounded result, if the exponent range were unbounded, exceeds that of the largest finite number of the specified result precision.

13.4.3.1 Action

The action to be taken depends on the setting of FPSCR[OE].

When Overflow exceptions re enabled (FPSCR[OE] = 1) and exponent overflow occurs, the following actions are taken:

- Overflow Exception is set

  FPSCR[OX] $\leftarrow 1$
- For double-precision arithmetic instructions, the exponent of the normalized intermediate result is adjusted by subtracting 1536.
- For single-precision arithmetic instructions and the **frsp** instruction, the exponent of the normalized intermediate result is adjusted by subtracting 192.
- FPR(FRT) $\leftarrow$ adjusted rounded result

- FPSCR[FPRF] ← class and sign of the result (± Normal Number)

When Overflow Exception is disabled (FPSCR[OE] = 0) and overflow occurs, the following actions are taken:

- Overflow Exception is set

  FPSCR[OX] ← 1
- Inexact Exception is set

  FPSCR[XX] ← 1
- The result is determined by the rounding mode (FPSCR[RN]) and the sign of the intermediate result as follows:
  - Round to Nearest
    Store ± Infinity, where the sign is the sign of the intermediate result
  - Round toward Zero
    Store the format's largest finite number with the sign of the intermediate result
  - Round toward +Infinity
    For negative overflow, store the format's most negative finite number; for positive overflow, store +Infinity
  - Round toward –Infinity
    For negative overflow, store – Infinity; for positive overflow, store the largest finite number of the format
- FPR(FRT) ← result
- FPSCR[FR] ← undefined
- FPSCR[FI] ← 1
- FPSCR[FPRF] ← class and sign of the result (± Infinity or ± Normal Number)

13.4.4 Underflow Exception

Underflow Exception is defined separately for the enabled and disabled states:

- Enabled:
  Underflow occurs when the intermediate result is "Tiny."
- Disabled:
  Underflow occurs when the intermediate result is "Tiny" and there is "Loss of Accuracy."

A "Tiny" result is detected before rounding, when a nonzero intermediate result computed as though both the precision and the exponent range were unbounded would be less in magnitude than the smallest normalized number.

If the intermediate result is "Tiny" and Underflow Exception is disabled (FPSCR[UE] = 0), the intermediate result is denormalized (See *Normalization and Denormalization* on page 168) and rounded (See *Rounding* on page 169) before being placed into the target FPR.

"Loss of Accuracy" is detected when the delivered result value differs from what would have been computed were both the precision and the exponent range unbounded.

13.4.4.1 Action

The action to be taken depends on the setting of FPSCR[UE].

When Underflow exception is enabled (FPSCR[UE] = 1) and exponent underflow occurs, the following actions are taken:

- Underflow Exception is set
  FPSCR[UX] ← 1
- For double-precision arithmetic instructions, the exponent of the normalized intermediate result is adjusted by adding 1536
- For single-precision arithmetic instructions and the **frsp** instruction, the exponent of the normalized intermediate result is adjusted by adding 192
- The adjusted rounded result is placed into the target FPR

  FPSCR[FPRF] ← class and sign of the result (± Normalized Number)

  Programming Note: The FR and FI bits are provided to allow the Enabled exception type Program interrupt, when taken because of an Underflow Exception, to simulate a "trap disabled" environment. That is, the FR and FI bits allow the Enabled exception type Program interrupt to unround the result, thus allowing the result to be denormalized.

When Underflow Exception is disabled (FPSCR[UE] = 0) and underflow occurs, the following actions are taken:

- Underflow Exception is set

  FPSCR[UX] ← 1
- FPR(FRT) ← rounded result
- FPSCR[FPRF] ← class and sign of the result (± Normalized Number, ± Denormalized Number, or ± Zero)

13.4.5 Inexact Exception

An Inexact Exception occurs when either of the following conditions occur during rounding:

- The rounded result differs from the intermediate result, assuming both the precision and the exponent range of the intermediate result to be unbounded. In this case, the result is said to be inexact. If the rounding causes an enabled Overflow Exception or an enabled Underflow Exception, an Inexact Exception also occurs only if the significands of the rounded result and the intermediate result differ.)
- The rounded result overflows and Overflow Exception is disabled.

13.4.5.1 Action

The action to be taken does not depend on the setting of FPSCR[XX].

When Inexact Exception occurs, the following actions are taken:

- Inexact Exception is set

  FPSCR[XX] ← 1
- FPR(FRT) ← rounded or overflowed result
- FPSCR[FPRF] ← class and sign of the result

  Programming Note: In some implementations, enabling Inexact Exceptions may degrade performance more than does enabling other types of floating-point exception.

13.5 Exception Priorities for Floating-Point Load and Store Instructions

The following prioritized list of exceptions may occur as a result of the attempted execution of any *Floating-Point Load* and *Store* instruction.

1. Debug (Instruction Address Compare)
2. Instruction TLB Error (all types)
3. Instruction Storage Interrupt (all types)
4. Program (Illegal Instruction)
5. Floating-Point Unavailable
6. Program (Unimplemented Operation)
7. Data TLB Error (all types)
8. Data Storage (all types)
9. Alignment
10. Debug (Data Address Compare, Data Value Compare)
11. Debug (Instruction Complete)

If an instruction causes both a Debug (Instruction Address Compare) exception, and a Debug (Data Address Compare) or Debug (Data Value Compare) exception, and does not cause any exception listed in items 2–9, both exceptions can be generated and recorded in the Debug Status Register (DBSR). A single Debug interrupt results.

13.6 Exception Priorities for other Floating-Point Instructions

The following prioritized list of exceptions may occur as a result of the attempted execution of any floating-point instruction other than a load or store.

1. Debug (Instruction Address Compare)
2. Instruction TLB Error (all types)
3. Instruction Storage Interrupt (all types)
4. Program (Illegal Instruction)
5. Floating-Point Unavailable
6. Program (Unimplemented Operation)
7. Program (Enabled)
8. Debug (Instruction Complete)

13.7 QNaN

If any of the source operands is a NaN, either signaling (SNaN) or quiet (QNaN), the result will be that NaN with the high-order fraction bit forced to 1 (that is, forced to a QNaN). The precedence, in decreasing order, is FRA, FRB, FRC. The resultant QNaN is only truncated on an **frsp[.]** instruction, in which case the most significant 35 bits are copied to the target, with the least significant 29 forced to zero.

Table 13-4. QNaN Result

R_a	R_b	R_c	Resultant QNaN[a]
NaN	X	X	R_a
—	—	X	R_b[b]
—	—	NaN	R_c

a. High-order fraction bit is forced to a 1
b. **frsp**: Result is $(FRB)_{0:11}$ || 1 || $(FRB)_{13:34}$|| $^{29}0$?

13.8 Updating FPRs on Exceptions

The target FPR is never updated on enabled invalid exceptions and enabled divide by zero exceptions. This requirement exists because an instruction may potentially use one of the source registers as a target register, yet it is necessary that the trap handler be able to examine and act upon the source operands.

In all other cases, a floating-point exception does not block the writing of the target FPR.

13.9 Floating-Point Status and Control Register

The computational instructions modify the FPSCR. With the exception of instructions which write directly to the FPSCR, none of the noncomputational instructions modify the FPSCR.

The FPSCR controls the handling of floating-point exceptions and records status resulting from the floating-point operations. $FPSCR_{0:23}$ are status bits. $FPSCR_{24:31}$ are control bits.

The exception bits in the FPSCR (bits 3:12, 21:23) are sticky; that is, once set to 1 they remain set to 1 until they are set to 0 by an **mcrfs**, **mtfsfi**, **mtfsf**, or **mtfsb0** instruction. The exception summary bits FPSCR[FX, FEX, VX] are not considered as "exception bits," and only FPSCR[FX] is sticky.

FPSCR[FEX, VX] are simply ORs of other FPSCR bits. Therefore, these bits are not listed among the FPSCR bits affected by the various instructions.

FPSCR[FPRF], which contains five result flag bits, is set for arithmetic, rounding, and conversion instructions based on the class of the result value placed into the target FPR. If any portion of a result is undefined, the value placed into FPSCR[FPRF] is undefined. *Table 13-5* describes how the values of the result flags in FPSCR[FPRF] correspond to the result value classes.

Table 13-5. FPSCR[FPRF] Result Flags

Result Flags					Result Value Class
C	<	>	=	?	
1	0	0	0	1	Quiet NaN
0	1	0	0	0	–Infinity
0	1	0	0	0	–Normalized Number
1	1	0	1	0	–Denormalized Number
1	0	0	1	0	–Zero
0	0	0	0	0	+Zero
1	0	1	0	0	+Denormalized Number
0	0	1	0	0	+Normalized Number
0	0	1	0	1	+Infinity

Figure 4-2 on page 161 illustrates the FPSCR.

13.10 Updating the CR

Architecturally, excepting floating-point instructions do not block the updating of the CR in the PPC440 processor. However, the PPC440 FPU blocks CR updates and requires software assistance to make them.

13.10.1 CR Fields

The CR fields are modified by various floating-point instructions.

Figure 13-1. Condition Register (CR)

0:3	CR0	Condition Register Field 0	
4:7	CR1	Condition Register Field 1	
8:11	CR2	Condition Register Field 2	
12:15	CR3	Condition Register Field 3	
16:19	CR4	Condition Register Field 4	
20:23	CR5	Condition Register Field 5	
24:27	CR6	Condition Register Field 6	
28:31	CR7	Condition Register Field 7	

13.10.2 Updating CR Fields

The floating-point compare instructions **fcmpo** and **fcmpu** specify a CR field that is updated with the compare results. update a field (specified by the instruction) of the CR.

Table 13-6 illustrates the bit encodings for a CR field containing the results of an **fcmpo** and **fcmpu** instruction.

Table 13-6. Bit Encodings for a CR Field

CR Field (Bit)	Description
0	*Floating-Point Less Than* (FL) Floating-point compare: (FRA) < (FRB)
1	*Floating-Point Greater Than* (FG) Floating-point compare: (FRA) > (FRB)
2	*Floating-Point Equal* (FE) Floating-point compare: (FRA) = (FRB)
3	*Floating-Point Unordered* (FU) Floating-point compare: One or both of (FRA) or (FRB) is a NaN.

The **mcrfs** instruction moves a specified FPSCR field into a CR field.

13.10.3 Generation of QNaN Results

If a disabled Invalid Operation exception is caused by operating on a NaN, the value returned follows the rules indicated in *Table 13-4* on page 279.

If the exception was not caused by operating on a NaN, a QNaN must be generated. The generated QNaN has a sign bit of 0, an exponent of all 1s, a high-order fraction bit of 1 with all other fraction bits of 0: 0x7FF8000000000000.

14. Reset and Initialization

This chapter describes the initial state of the PPC460EX/EXr/GT after a hardware reset, and contains a description of the initialization software required to complete initialization so that the PPC460EX/EXr/GT can begin executing application code. Initialization of other on-chip and off-chip system components is described in the appropriate chapter.

14.1 Reset Signals

The PPC460EX/EXr/GT provides four reset signals, SysReset, HISRRst, PCI0Reset, and ExtReset. SysReset and HISRRst are inputs and PCI0Reset and ExtReset are outputs.

When the SysReset signal is asserted by an off-chip device, such as during power-on-reset (POR), the chip responds by performing a system reset as described in a following section.

HISRRst is the hardware-initiated self refresh and reset input. When it is active, it requests the DDR SDRAM controller to put the memory in self-refresh mode. When self refresh is acknowledged, the reset logic performs a system reset.

While the chip is performing a system reset, the ExtReset output signal is asserted. If the chip is configured for PCI host mode, then the PCI0Reset output signal is also asserted. The duration of the internal reset process is at least 16384 SysClk cycles. This enables the PPC460EX/GT to reset itself and other attached devices using ExtReset or PCI0Reset to drive their reset inputs.

The ExtReset signal is used by synchronous peripheral devices served by the external bus clock, such as ROM and external masters. During chip and system resets, ExtReset is asserted until the PerClk signal is stable and all internal resets are released. Its duration in effect is the same as that of signal RstTopAllLogicRst which resets all on-chip peripherals. See *Figure 14-2*.

The PCI0Reset signal is used to drive the PCI RST# input of attached PCI devices when the PPC460EX/GT is configured to be in PCI Host mode. The timing of PCI0Reset is similar to that of ExtReset. When the PPC460EX/GT is configured to be in PCI adapter mode, then PCI0Reset is not used. SysReset should be connected to the PCI RST# signal from the external PCI host.

14.2 Reset Types

Three types of reset, each with different scope, are possible in the PPC460EX/EXr/GT. A CPU reset affects only the processor and the FPU. Chip resets affect the processor, FPU, and all on-chip peripherals. System resets affect the processor, FPU, all on-chip peripherals, and any off-chip devices connected to the PPC460EX/EXr/GT ExtReset signal. The effects of system, chip and CPU resets on the processor and FPU are identical. To determine which reset type occurred, the most-recent reset (MRR) field of the Debug Status Register (DBSR) can be examined.

14.2.1 CPU Reset

A CPU reset results in a reset of the processor and FPU. No other on-chip logic is affected. Reset_top logic, outside the processor, detects the CPU reset request and asserts the reset input to the processor and FPU. The duration of a CPU reset will last 10 system clock cycles beyond the CPU reset request, which typically lasts 4 CPU clock cycles.

Note: Set DBCR[RST]=0b01 to generate a CPU reset.

14.2.2 Chip Reset

A chip reset results in the reset of the processor, FPU, and on-chip peripherals. A chip reset request is generated by the processor so that the processor, FPU, and all on-chip peripherals reset for an extended period while the CPR PLL is allowed to re-lock. The duration of a chip reset will last 10 system clocks beyond the CPR PLL locking.

During chip reset, the $\overline{\text{ExtReset}}$ signal is driven low to ensure the reset of synchronous devices that use the external bus clock signal, PerClk.

Note: Set DBCR[RST]=0b10 to generate a Chip Reset.

14.2.3 System Reset

A system reset results in a reset of all PPC460EX/EXr/GT logic, including the processor, FPU, internal phase-locked loops (CPR PLL), and on-chip peripherals. A system reset can be initiated externally or internally. External system resets are initiated by the assertion of the $\overline{\text{SysReset}}$ signal for at least 32 SysClk cycles. Internal system resets are initiated by either the processor or the PCI power management logic. The duration of the system reset operation internal to the chip is 10 system clocks beyond the CPR PLL locking.

When a system reset is requested internally, $\overline{\text{ExtReset}}$ signal is asserted to enable other attached devices to be reset at the same time. The internal reset and $\overline{\text{ExtReset}}$ signal are asserted for 16384 SysClk cycles. During this time all internal clocks and PerClk clock toggle.

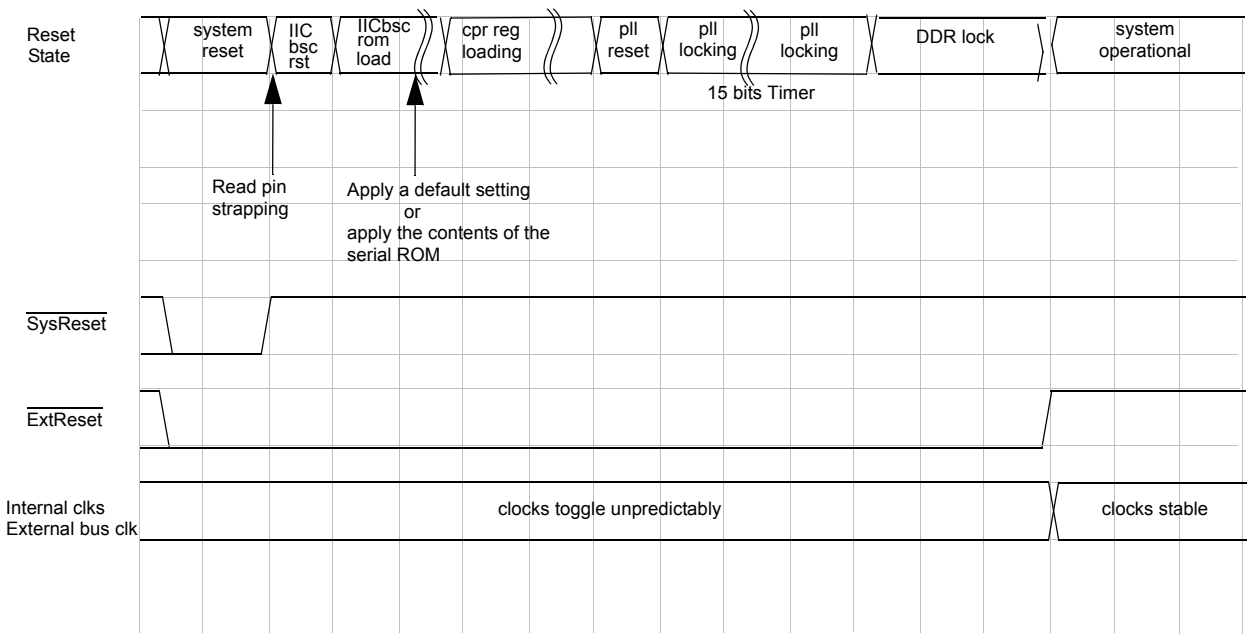
After the $\overline{\text{SysReset}}$ signal is deasserted, the CPR PLL begins its locking process, which requires about 10,000 SysClk cycles when local feedback is selected and 20,000 SysClk cycles when remote PerClk feedback is selected. During this time all internal clocks and PerClk clock toggle.

When the PLL locking process is complete, internal resets are released by Reset_top logic, and the processor begins its initial instruction fetch.

During system reset, the $\overline{\text{ExtReset}}$ signal is driven low to ensure the reset of synchronous devices that use the external bus clock signal, PerClk.

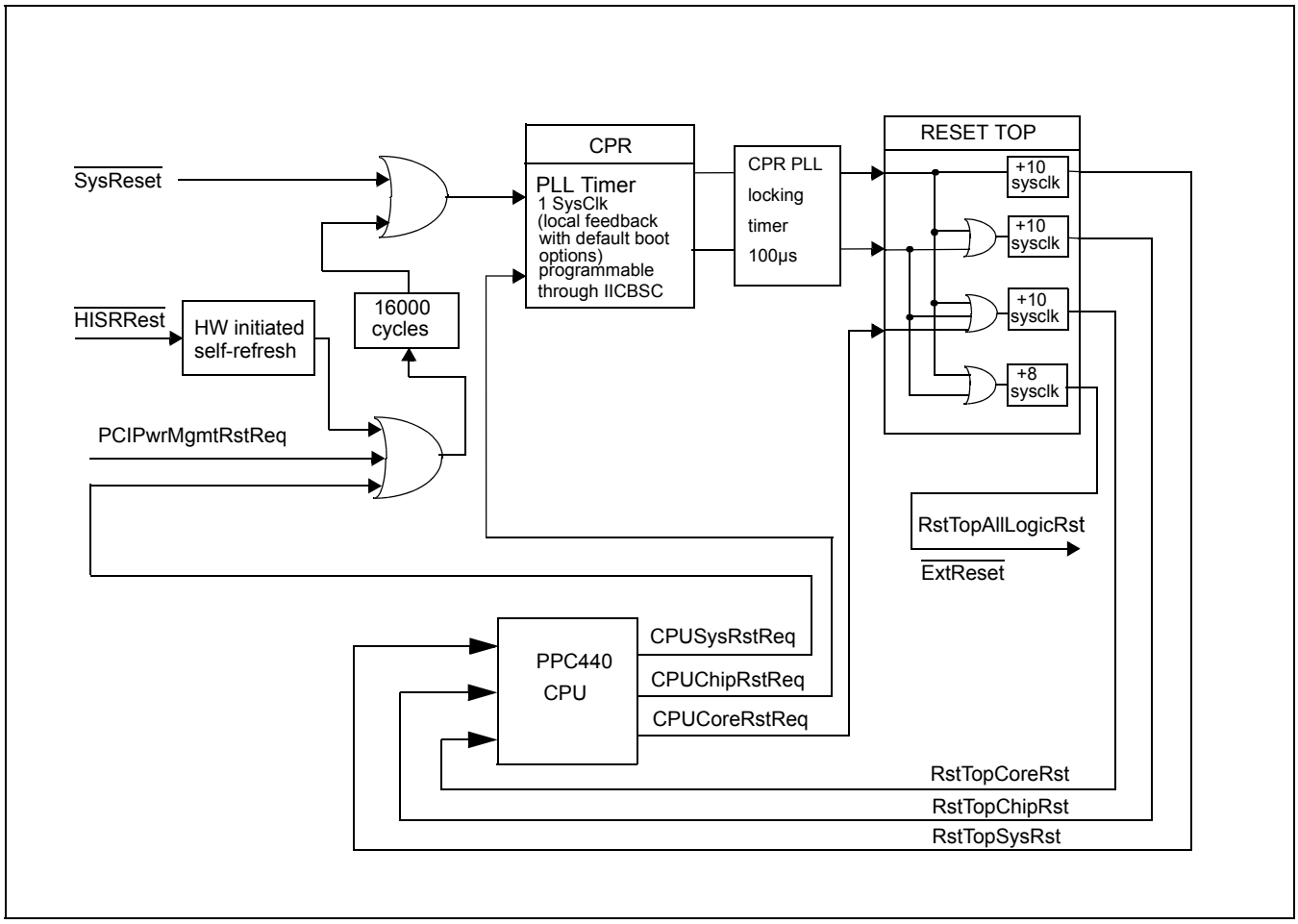
Note: Software can generate a System Reset by setting DBCR[RST]=0b11.

Figure 14-1. PPC460EX/EXr/GT Power-on Reset Process



Note: The boot memory is accessed shortly after $\overline{\text{ExtReset}}$ is driven high following a Chip or System reset. To ensure the boot memory has completed reset prior to the first access, use $\overline{\text{SysReset}}$ as the reset input to the boot memory.

Figure 14-2. PPC460EX/EXr/GT Reset



14.3 Processor Initiated Resets

The PPC460EX/EXr/GT processor can initiate three types of processor resets: core, chip, and system. Each type of reset can be generated by a JTAG debug tool, by the second expiration of the watchdog timer, or by writing a non-zero value to the Reset (RST) field of Debug Control Register 0 (DBCR0).

The reset type is recorded in two status registers, DBSR and TSR. The Debug Status Register MRR bit field (DBSR[MRR]) records most-recent reset. The Timer Status Register WRS bit field (TSR[WRS]) records the reset type of the most recent watchdog reset.

14.4 Processor State After Reset

Generally, register contents and the status of other facilities within the processor are unpredictable following a hardware reset. Reset initializes only the resources required to fetch and execute instructions from the initial program memory page. This ensures that after a hardware reset, the processor state will always be the same (provided that the correct software initialization sequence is performed). System software must completely configure the remaining PPC460EX/EXr/GT resources, as well as the other facilities within the chip and the system.

Note: Refer to the *PPC440H6 Processor User's Manual* for details on all of the registers mentioned in this section.

The following list summarizes the requirements of the Book-E Enhanced PowerPC Architecture with regard to the processor state after reset, before any additional initialization by software.

- All fields of the Machine State Register (MSR) must be set to 0, disabling all asynchronous interrupts, placing the processor in supervisor mode, and specifying that instruction and data accesses are to the system (as opposed to application) address space.
- DBCR0[RST] must be set to 0, thus ending any previous software-initiated reset operation.
- DBSR[MRR] must identify the type of reset operation just completed (core, chip, or HISRRst). See *Reset Types* on page 283 for details.
- TCR[WRC] must be set to 0, thus disabling the Watchdog timer reset operation.
- TSR[WRS] must identify the type of reset operation just ended (if the reset was initiated by the Watchdog Timer). Otherwise TSR[WRS] is unchanged from its pre-reset value).
- The PVR must be defined (after reset and otherwise) to contain a value that indicates the specific processor implementation.
- The program counter (PC) must be set to 0xFFFFFFFC, the effective address (EA) of the last word of the address space.

The memory management resources are set to values such that the processor is able to successfully fetch and execute instructions and read (but not write) data within the 4KB program memory page located at the end of the 32-bit effective address space. Exactly how this is accomplished is implementation-dependent. For example, it may be that a TLB entry has been established in a manner that is visible to software using the TLB management instructions. Regardless of how the implementation enables access to the initial program memory page, instruction execution starts at the effective address of 0xFFFFFFFC (the last word of the effective address space). The instruction at this address must be an unconditional branch to the start of the initialization sequence, which must lie somewhere within the initial 4KB program memory page. The real address to which the initial effective address will be translated is also implementation- or system-dependent, as are the various storage attributes of the initial program memory page such as the caching inhibited and endian attributes.

Note: In the PPC460EX/EXr/GT, a single entry is established in the instruction shadow TLB (ITLB) and data shadow TLB (DTLB) when reset is performed, initialized with the values shown in the TLB Entry section of *Table 14-1 Reset Values of Registers and Other PPC460EX/EXr/GT Facilities* on page 287. Initialization software *must* insert an entry into the UTLB for this same memory region before performing any context-synchronizing operation (including causing any exceptions that would result in an interrupt). (Context-synchronizing operations invalidate the shadow TLB entries.)

Table 14-1. Reset Values of Registers and Other PPC460EX/EXr/GT Facilities

Resource	Field	Reset Value	Comment
CCR0	DAPUIB	0	Enable broadcast of instruction data to auxiliary processor interface
	DTB	0	Enable broadcast of trace information

Table 14-1. Reset Values of Registers and Other PPC460EX/EXr/GT Facilities (Continued)

Resource	Field	Reset Value	Comment
CCR1	ICDPEI	0	Disable Parity Error Insertion (enabled only for s/w testing)
	ICTPEI	0	
	DCTPEI	0	
	DCDPEI	0	
	DCUPEI	0	
	DCMPEI	0	
	FCOM	0	Do not force cache ops to miss.
	MMUPEI	0	Disable Parity Error Insertion (enabled only for s/w testing)
	FFF	0	Flush only as much data from dirty lines as needed.
DBCR0	EDM	0	External Debug mode disabled
	RST	0b00	Software-initiated debug reset disabled
	ICMP	0	Instruction completion debug events disabled
	BRT	0	Branch taken debug events disabled
	IAC1	0	Instruction Address Compare 1 (IAC1) debug events disabled
	IAC2	0	IAC2 debug events disabled
	IAC3	0	IAC3 debug events disabled
	IAC4	0	IAC4 debug events disabled
DBSR	UDE	0	Unconditional debug event has not occurred
	MRR	Reset-dependent	Indicates most recent type of reset as follows: 00 No reset has occurred since this field last cleared by software 01 Core reset 10 Chip reset 11 System reset
	ICMP	0	Instruction completion debug event has not occurred
	BRT	0	Branch taken debug event has not occurred
	IRPT	0	Interrupt debug event has not occurred
	TRAP	0	Trap debug event has not occurred
	IAC1	0	IAC1 debug event has not occurred
	IAC2	0	IAC2 debug event has not occurred
	IAC3	0	IAC3 debug event has not occurred
	IAC4	0	IAC4 debug event has not occurred
	DAC1R	0	Data address compare 1 (DAC1) read debug event has not occurred
	DAC1W	0	DAC1 write debug event has not occurred
	DAC2R	0	DAC2 read debug event has not occurred
	DAC2W	0	DAC2 write debug event has not occurred
	RET	0	Return debug event has not occurred
ESR	MCI	0	Synchronous Instruction Machine Check exception has not occurred
MCSR	MCS	0	Asynchronous Instruction Machine Check exception has not occurred

Table 14-1. Reset Values of Registers and Other PPC460EX/EXr/GT Facilities (Continued)

Resource	Field	Reset Value	Comment
MSR	WE	0	Processor is not in wait state.
	CE	0	Asynchronous critical interrupts disabled
	EE	0	Asynchronous non-critical interrupts disabled
	PR	0	Processor in supervisor mode
	ME	0	Machine Check interrupts disabled
	DWE	0	Debug Wait mode disabled
	DE	0	Debug interrupts disabled
	IS	0	Instruction fetch access is to system-level virtual address space
	DS	0	Data access is to system level virtual address space
PC		0xFFFFFFFC	Initial reset instruction fetched from last word of effective address space
PVR	0:31	System dependent	Refer to *PPC460EX/EXr/GT Embedded Processor Data Sheet* for the PVR value
RSTCFG	U0	0	The U0 storage attribute has no effect in the PPC460EX/EXr/GT.
	U1	0	Memory page contain normal instructions or data
	U2	0	Storage misses do not cause a line to allocated in the data cache
	U3	0	The U3 storage attribute has no effect in the PPC460EX/EXr/GT.
	E	0	Memory pages are big endian
	ERPN	System dependent	If ROM is connected to EBC, ERPN is 0b0001. If ROM is connected to PCI, ERPN is 0b0010.
TCR	WRC	0b00	Watchdog Timer reset disabled
TLBentry[1]	$EPN_{0:19}$	0xFFFFF000	Match EA of initial reset instruction ($EPN_{20:21}$ are undefined, as they are not compared to the EA because the page size is 4KB).
	V	1	Translation table entry for the initial program memory page is valid.
	TS	0	Initial program memory page is in system-level virtual address space.
	SIZE	0b0001	Initial program memory page size is 4KB.
	TID	0x00	Initial program memory page is globally shared; no match required against PID register.
	$RPN_{0:21}$	0xFFFFF || 0b00	Initial program memory page mapped effective = real.
	ERPN	0b0100 0b1100	Extended real page number of the initial program memory page is specified by core input signals. Shadow TLB entry at reset is 0x4 FF00 0000 if PPC460EX/EXr/GT is strapped to boot from ROM connected to the EBC and 0xC FF00 0000 if PPC460EX/EXr/GT is strapped to boot from PCI memory.
	U0	0	The U0 storage attribute has no effect in the PPC460EX/EXr/GT.
	U1	0	Memory page contains normal instructions or data.
	U2	0	Storage misses do not cause a line to be allocated in the data cache.
	U3	0	The U3 storage attribute has no effect in the PPC460EX/EXr/GT.
	W	0	Write-through storage attribute disabled.
	I	1	Caching inhibited storage attribute enabled.
	M	0	Memory coherent storage attribute disabled.
	G	1	Guarded storage attribute enabled.
	E	0	Memory pages are big endian.
	SX	1	Supervisor mode execution access enabled.
	SW	0	Supervisor mode write access disabled.
	SR	1	Supervisor mode read access enabled.

Table 14-1. Reset Values of Registers and Other PPC460EX/EXr/GT Facilities (Continued)

Resource	Field	Reset Value	Comment
TSR	WRS	Copy of TCR[WRC]	If reset caused by Watchdog Timer
		Unchanged	If reset not caused by Watchdog Timer
		Undefined	After power-up
Note: "TLBentry" refers to an entry in the shadow instruction and data TLB arrays that is automatically configured by the PPC460EX/EXr/GT to enable fetching and reading (but not writing) from the initial program memory page. This entry is not architecturally visible to software, and is invalidated upon any context-synchronizing operation. Software must initialize a corresponding entry in the main unified TLB array before executing any operation which could lead to a context synchronization. See *Initialization Software Requirements* on page 300 for more information.			

14.5 PPC460EX/EXr/GT Chip Initialization

The chip configuration registers fall under two categories:

1. Clocking and power-on reset (CPR0) registers which are accessed indirectly through the CPR0_CFGADDR and CPR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions.
2. System device registers (SDR0) which are accessed indirectly through the SDR0_CFGADDR and SDR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions.

During PPC460EX/EXr/GT initialization, some chip control registers must be initialized to ensure proper chip operation. Peripheral devices are also initialized as appropriate for the system design. Registers that control chip clocking and power-on reset are described in *Clocking* on page 237 while registers that control chip pin strapping are described in *Bootstrap Operations* on page 213. Initialization for peripheral devices is described in individual chapters as needed including chip level serial device control registers. The section that follows, lists registers that control miscellaneous chip devices that are not discussed in any other chapter.

Table 14-2. Initialization SDRs

Mnemonic	Register	Offset	Access	Page
SDR0_ECID0	Electronic Chip ID 0	0x0080	Read only	291
SDR0_ECID1	Electronic Chip ID 1	0x0081	Read only	291
SDR0_ECID2	Electronic Chip ID 2	0x0082	Read only	291
SDR0_ECID3	Electronic Chip ID 3	0x0083	Read only	291
SDR0_JTAG	JTAG ID	0x00C0	Read only	291
SDR0_PFC0	Pin Function Control 0	0x4100	R/W	292
SDR0_PFC1	Pin Function Control 1	0x4101	R/W	292
SDR0_ETH_PLL	Ethernet PLL Configuration	0x4102	R/W	293
SDR0_ETH_CFG	Ethernet Configuration	0x4103	R/W	293
SDR0_ETH_STS	Ethernet Status	0x4104	R/W	295
SDR0_SLPIPE	PLB Slave Address Pipeline Disabling	0x0220	R/W	296
SDR0_SRST0	Soft Reset Register 0	0x0200	R/W	296
SDR0_SRST1	Soft Reset Register 1	0x0201	R/W	297
SDR0_MFR	Miscellaneous Function Register	0x4300	R/W	298

14.5.1 Initialization SDRs

All SDRs are accessed with the move-to-DCR (**mtdcr**) and move-from-DCR (**mfdcr**) instructions using indirect addressing. In indirect addressing, the **mtdcr** and **mfdcr** instructions access a given SDR by means of its address offset which is contained in the SDR0_CFGADDR register. The data for the register specified in SDR0_CFGADDR is moved to or moved from the SDR0_CFGDATA register. The SDR address and data registers are shown in *System DCR Configuration Registers* on page 154.

14.5.1.1 Electronic Chip ID Register 0 (SDR0_ECID0)

Figure 14-3. Electronic Chip ID Register 0 (SDR0_ECID0)

0:31	ECID0	Electronic chip ID 0	Bits 0:31 of 112-bit ID

14.5.1.2 Electronic Chip ID Register 1 (SDR0_ECID1)

Figure 14-4. Electronic Chip ID Register 1 (SDR0_ECID1)

0:31	ECID1	Electronic chip ID 1	Bits 32:63 of 112-bit ID

14.5.1.3 Electronic Chip ID Register 2 (SDR0_ECID2)

Figure 14-5. Electronic Chip ID Register 2 (SDR0_ECID2)

0:31	ECID2	Electronic chip ID 2	Bits 64:95 of 112-bit ID

14.5.1.4 Electronic Chip ID Register 3 (SDR0_ECID3)

Figure 14-6. Electronic Chip ID Register 3 (SDR0_ECID3)

0:15	ECID3	Electronic chip ID 3	Bits 96:111 of 112-bit ID
16:31		Reserved	

14.5.1.5 JTAG ID Register (SDR0_JTAG)

Figure 14-7. JTAG ID Register (SDR0_JTAG)

0:31	JTAG	JTAG ID	JTAG = 0x144101E1

14.5.1.6 Pin Function Control Register 0 (SDR0_PFC0)

Figure 14-8. Pin Function Control Register 0 (SDR0_PFC0)

0:15		Reserved	
16	DBG	Debug enable 0 Functional 1 CPU trace	
17:31	GnnE	GPIO nn output 0 Disable 1 Enable	nn = 49–63 (for example nn = 49 for bit 17, nn = 50 for bit 18, and so on). **Note:** IF CPU trace is enabled, it has highest priority (CPU trace, then GPIO).

14.5.1.7 Pin Function Control Register 1 (SDR0_PFC1)

Figure 14-9. Pin Function Control Register 1 (SDR0_PFC1)

0:5		Reserved	
6	U1ME	UART1 Mode Enable Selects the function of UART1 signals in 4-pin mode. 0 Enable $\overline{\text{UART1DSR}}$/$\overline{\text{CTS}}$ as DSR and $\overline{\text{UART1DTR}}$/$\overline{\text{RTS}}$ as $\overline{\text{DTR}}$ 1 Enable $\overline{\text{UART1DSR}}$/$\overline{\text{CTS}}$ as CTS and $\overline{\text{UART1DTR}}$/$\overline{\text{RTS}}$ as RTS	See multiplexed signals in the PPC460EX/EXr/GT data sheet and pin sharing tables in the GPIO chapter. **Note:** U0IM must be set to 1 for 4-pin mode.
7:11		Reserved	
12	U0ME	UART0 Mode Enable Selects the function of UART0 signals in 4-pin mode. 0 Enable $\overline{\text{UART0DSR}}$/$\overline{\text{CTS}}$ as DSR and $\overline{\text{UART0DTR}}$/$\overline{\text{RTS}}$ as $\overline{\text{DTR}}$ 1 Enable $\overline{\text{UART0DSR}}$/$\overline{\text{CTS}}$ as CTS and $\overline{\text{UART0DTR}}$/$\overline{\text{RTS}}$ as RTS	See multiplexed signals in the PPC460EX/EXr/GT data sheet and pin sharing tables in the GPIO chapter. **Note:** U0ME must be set to 1 when U0IM = 0.
13	U0IM	UART0 Interface Mode 0 8 pins 1 4 pins	
14	SIS	SPI/IIC1 Selection 0 SPI enabled 1 IIC1 enabled	SIS controls the function of signals: [IIC1SClk]SCPClkOut [IIC1SData]SCPDI
15:31		Reserved	

14.5.1.8 Ethernet PLL Configuration Register (SDR0_ETH_PLL)

The Ethernet PLL generates two fixed frequencies for the Ethernet Subsystem: 1) 1250MHz for SGMII, and 2) 125MHz for all other modes. Depending on the mode, these clocks might be further divided for use by the logic. The Ethernet reference clock is required to be 125MHz, and the VCO of the PLL runs at 1250MHz. The PLL is hard wired to use local feedback.

Figure 14-10. Ethernet PLL Configuration Register (SDR0_ETH_PLL)

0	PLLLOCK	Ethernet PLL lock indication	Read-only.
1:3		Reserved	
4	BYPASS	Bypass mode enable 0 Normal functional mode 1 PLLOUTA, PLLOUTB/C outputs are buffered versions of the Ethernet reference clock	
5	STOPCLK	Output clock disable 0 PLLOUTA, PLLOUTB/C outputs enabled 1 PLLOUTA, PLLOUTB/C outputs high	
6:15	TUNE	Loop stability tuning bits	Local Feedback: M = Fbk-Div = 10 Reset value = 0b1101000000.
16:23	MULTI	Frequency multiplication selector	Feedback Divider = 10 Reset value = 0b11010100.
24:27	RANGEB	PLLOUTB/C frequency selector	GMII Clock Forward Divider = 10 Reset value = 0b1100.
28:31	RANGEA	PLLOUTA frequency selector	SGMII Clock Forward Divider = 1 Reset value = 0b0000.

14.5.1.9 Ethernet Configuration Register (SDR0_ETH_CFG)

Figure 14-11. Ethernet Configuration Register (SDR0_ETH_CFG)

0:1	SGMII0_SPEED_SEL	SGMII0 Speed Select 00 1Gbps 01 reserved 10 100 Mbps 11 10Mpbs	Forces the speed of the SGMII link for EMAC0 to 1Gbps, 100Mbps, or 10Mbps. If auto-negotiation is required, set SGMII0_SPEED_SEL=00. (SGMII_AUTO only applies to PPC460EX/EXr/GT rev B. See errata for PPC460EX/EXr/GT rev A.)
2:3		Reserved	
4:5	SGMII1_SPEED_SEL	SGMII1 Speed Select 00 1Gbps 01 reserved 10 100 Mbps 11 10Mpbs	Forces the speed of the SGMII link for EMAC1 to 1Gbps, 100Mbps, or 10Mbps. If auto-negotiation is required, set SGMII1_SPEED_SEL=00. (SGMII_AUTO only applies to PPC460EX/EXr/GT rev B. See errata for PPC460EX/EXr/GT rev A.)
6:7		Reserved	

8	SGMII_AUTO	SGMII Auto-negotiation enable 0 SGMII Auto-negotiation disabled 1 SGMII Auto-negotiation enabled	SGMII_AUTO when set to 1 enables auto-negotiation for SGMII0 and SGMII1 ports. (SGMII_AUTO only applies to PPC460EX/EXr/GT rev B. See errata for PPC460EX/EXr/GT rev A.) **Note:** SGMII2 port on PPC460GT does not support auto-negotiation. SGMII2 only operates at 1Gbps.
9	SGMII2_LPBK	SGMII2 Port loopback enable 0 SGMII2 Port loopback disabled 1 SGMII2 Port loopback enabled	PPC460GT only When SGMII 2 port is enabled, transmit data is looped back internally to receive data.
10	SGMII1_LPBK	SGMII1 Port loopback enable 0 SGMII1 Port loopback disabled 1 SGMII1 Port loopback enabled	When SGMII 1 port is enabled, transmit data is looped back internally to receive data.
11	SGMII0_LPBK	SGMII0 Port loopback enable 0 SGMII0 Port loopback disabled 1 SGMII0 Port loopback enabled	PPC460EX and PPC460GT only When SGMII 0 port is enabled, transmit data is looped back internally to receive data.
12		Reserved	
13	SGMII2_ENABLE	SGMII2 Port enable 0 SGMII2 Port disabled 1 SGMII2 Port enabled	PPC460GT only When SGMII 2 port is enabled, EMAC2 cannot be used for the GMC 1 port.
14	SGMII1_ENABLE	SGMII1 Port enable 0 SGMII1 Port disabled 1 SGMII1 Port enabled	When SGMII 1 port is enabled, EMAC1 cannot be used for the GMC 0 port.
15	SGMII0_ENABLE	SGMII0 Port enable 0 SGMII0 Port disabled 1 SGMII0 Port enabled	PPC460EX and PPC460GT only When SGMII 0 port is enabled, EMAC0 cannot be used for the GMC 0 port.
16:17		Reserved	
18	TAHOE1_BYPASS	TAHOE1 Bypass selector 0 No Bypass 1 Bypass	When TAHOE1 is bypassed, MAL is connected to EMAC1.
19	TAHOE0_BYPASS	TAHOE0 Bypass selector 0 No Bypass 1 Bypass	When TAHOE0 is bypassed, MAL is connected to EMAC0.
20	EMAC3_PHY_CLK_SEL	EMAC3 PHY clock selector 0 Clock from PHY 1 Internal LOOP Clock	PPC460GT only
21	EMAC2_PHY_CLK_SEL	EMAC2 PHY clock selector 0 Clock from PHY 1 Internal LOOP Clock	PPC460GT only
22	EMAC1_PHY_CLK_SEL	EMAC1 PHY clock selector 0 Clock from PHY 1 Internal LOOP Clock	
23	EMAC0_PHY_CLK_SEL	EMAC0 PHY clock selector 0 Clock from PHY 1 Internal LOOP Clock	
24:25		Reserved	
26:27	MDIO_SEL	MDIO source selector 00 EMAC0 01 Reserved 10 Reserved 11 Reserved	

28:29	ZMIIM	ZMII bridge mode selector 00 MII 01 Reserved 10 Reserved 11 Reserved	Reset value = SDR0_SDSTP1[ZMIIM]
30	GMC1BS	GMC Port 1 bridge selector 0 RGMII1 Bridge 1 Reserved	PPC460GT only Reset value = SDR0_SDSTP1[GMC1BS]
31	GMC0BS	GMC Port 0 bridge selector 0 RGMII0 Bridge 1 ZMII Bridge	Reset value = SDR0_SDSTP1[GMC0BS]

14.5.1.10 Ethernet Status Register (SDR0_ETH_STS)

For all fields in the register, write 1 to clear.

Figure 14-12. Ethernet Status Register (SDR0_ETH_STS)

0:20		Reserved	
21	SGMII2_TX_LOC	SGMII2 Transmit Loss of Clock	PPC460GT only
22	SGMII1_TX_LOC	SGMII1 Transmit Loss of Clock	
23	SGMII0_TX_LOC	SGMII0 Transmit Loss of Clock	PPC460EX and PPC460GT only
24		Reserved	
25	SGMII2_RX_LOS	SGMII2 Receive Loss of Signal	PPC460GT only
26	SGMII1_RX_LOS	SGMII1 Receive Loss of Signal	
27	SGMII0_RX_LOS	SGMII0 Receive Loss of Signal	PPC460EX and PPC460GT only
28		Reserved	
29	SGMII2_RX_LOC	SGMII2 Receive Loss of Clock	PPC460GT only
30	SGMII1_RX_LOC	SGMII1 Receive Loss of Clock	
31	SGMII0_RX_LOC	SGMII0 Receive Loss of Clock	PPC460EX and PPC460GT only

14.5.1.11 Slave Address Pipeline Register (SDR0_SLPIPE)

For all fields in the register, the Reset value = 0xFE000000.

Figure 14-13. Slave Address Pipeline Register (SDR0_SLPIPE)

0	SRAP	SRAM Address Pipeline 0 Disabled 1 Enabled	
1	DSHBAP	DDR SDRAM HB Address Pipeline 0 Disabled 1 Enabled	
2	POBAP	PLB to OPB Bridge Address Pipeline 0 Disabled 1 Enabled	
3	DMAAP	I2O DMA Address Pipeline 0 Disabled 1 Enabled	
4	DSLLAP	DDR SDRAM LL Address Pipeline 0 Disabled 1 Enabled	
5	PCI0AP	PCI0 Address Pipeline 0 Disabled 1 Enabled	
6	OCMAP	OCM Address Pipeline 0 Disabled 1 Enabled	
7:31		Reserved	

14.5.1.12 Soft Reset Register 0 (SDR0_SRST0)

Figure 14-14. Soft Reset Register 0 (SDR0_SRST0)

0	BGO	PLB4 to OPB bridge	
1	PLB4	PLB4 arbiter	
2	EBC	External bus controller	
3	OPB	OPB arbiter	
4	UART0	Universal asynchronous receiver/transmitter 0	
5	UART1	Universal asynchronous receiver/transmitter 1	
6	IIC0	Inter integrated circuit 0	
7	IIC1	Inter integrated circuit 1	
8	GPIO0	General purpose I/O	
9	GPT	General purpose timer	
10	DMC	DDR1/2 SDRAM memory controller	
11	PCI0	PCI0	
12:13		Reserved	
14	CPM	Clock and power management	

15	I2ODMA	I2O DMA	
16	UIC0	Universal Interrupt Controller 0	
17	UIC1	Universal Interrupt Controller 1	
18	SRAM	Internal SRAM controller	
19	UIC2	Universal Interrupt Controller 2	
20	UIC3	Universal Interrupt Controller 3	
21	OCM	On-Chip Memory	
22	UART2	UART2	
23	MAL	Media access layer	
24		Reserved	
25	GPTR	General purpose timer	Reset the Time Base counter only.
26:28		Reserved	
29	L2C	L2 cache controller	
30	UART3	UART3	
31	GPIO1	General purpose I/O 1	

14.5.1.13 Soft Reset Register 1 (SDR0_SRST1)

Figure 14-15. Soft Reset Register 1 (SDR0_SRST1)

0	RLL	SRIO RLL	PPC460GT only
1	SCP	Serial communications port	
2	PLBARB	PLB Arbiter	
3	EIPPKP	EIPPKP	
4	EIP94	EIP-94	
5	EMAC0	Ethernet media access controller 0	
6	EMAC1	Ethernet media access controller 1	
7	EMAC2	Ethernet media access controller 2	PPC460GT only
8	EMAC3	Ethernet media access controller 3	PPC460GT only
9	ZMII	Ethernet ZMII	
10	RGMII0	Ethernet RGMII 0	
11	RGMII1	Ethernet RGMII 1	PPC460GT only
12	DMA2P40	DMA to PLB4	
13	DMACH	DMA Channel to PLB4	
14	SATAPHY	Serial ATA PHY	PPC460EX and PPC460EXr only
15	SRIODEV	Serial Rapid IO core, PCS, and SerDes	PPC460GT only
16	SRIOPCS	Serial Rapid IO core and PCS	PPC460GT only

17	NDFC	NAND flash controller	
18	SRIOPLB	Serial Rapid IO PLB	PPC460GT only
19	ETHPLL	Ethernet PLL	
20	TAHOE1	Ethernet Tahoe 1	
21	TAHOE0	Ethernet Tahoe 0	
22	SGMII0	Ethernet SGMII 0	PPC460EX and PPC460GT only
23	SGMII1	Ethernet SGMII 1	
24	SGMII2	Ethernet SGMII 2	PPC460GT only
25	AHB	PLB4XAHB bridge	PPC460EX and PPC460EXr only Reset value = 1 (reset).
26	USBOTGPHY	USB 2.0 OTG PHY	PPC460EX and PPC460EXr only
27	USBOTG	USB 2.0 OTG controller	PPC460EX and PPC460EXr only
28	USBHOST	USB 2.0 Host controller	PPC460EX and PPC460EXr only
29	AHBDMAC	AHB DMA controller	PPC460EX and PPC460EXr only
30	AHBICM	AHB inter-connect matrix	PPC460EX and PPC460EXr only
31	SATA	Serial ATA controller	PPC460EX and PPC460EXr only

14.5.1.14 Miscellaneous Function Register (SDR0_MFR)

Figure 14-16. Miscellaneous Function Register (SDR0_MFR)

0:2		Reserved	
3	PNC	PCI Prefetch New Context 0 Buffer validity not affected by read 1 Buffer invalidated after read completion	Enables read from "special purpose read prefetch buffer" to mark it invalid following the completion of returning the read data from that buffer.
4	HDF	HSDMA Disable Fix 0 Fix not disabled 1 Fix disabled	Disables fix for HSDMA hang in CHECK_ERROR state.
5	PMRC	PCI to I2O Message Routing Control 0 Messages routed between PCI0 and I2ODMA 1 Messages routed between PIH and I2ODMA	
6	ZIE	ZMII Zero Idle Enable 0 Idle cycle between packets can be all zeros or all ones 1 Idle cycle between packets must be all zeros	
7		Reserved	
8	T0TxFI	Parity Error TAHOE0 Tx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into TAHOE0 Tx FIFO	
9	T0TxFH	Parity Error TAHOE0 Tx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into TAHOE0 Tx FIFO	
10	T1TxFI	Parity Error TAHOE1 Tx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into TAHOE1 Tx FIFO	
11	T1TxFH	Parity Error TAHOE1 Tx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into TAHOE1 Tx FIFO	

12:15		Reserved	
16	E0TxFI	Parity Error EMAC0 Tx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC0 Tx FIFO	
17	E0TxFH	Parity Error EMAC0 Tx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC0 Tx FIFO	
18	E0RxFI	Parity Error EMAC0 Rx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC0 Rx FIFO	
19	E0RxFH	Parity Error EMAC0 Rx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC0 Rx Fifo	
20	E1TxFI	Parity Error EMAC1 Tx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC1 Tx FIFO	
21	E1TxFH	Parity Error EMAC1 Tx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC1 Tx FIFO	
22	E1RxFI	Parity Error EMAC1 Rx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC1 Rx FIFO	
23	E1RxFH	Parity Error EMAC1 Rx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC1 Rx FIFO	
24	E2TxFI	Parity Error EMAC2 Tx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC2 Tx FIFO	PPC460GT only
25	E2TxFH	Parity Error EMAC2 Tx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC2 Tx FIFO	PPC460GT only
26	E2RxFI	Parity Error EMAC2 Rx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC2 Rx FIFO	PPC460GT only
27	E2RxFH	Parity Error EMAC2 Rx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC2 Rx Fifo	PPC460GT only
28	E3TxFI	Parity Error EMAC3 Tx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC3 Tx FIFO	PPC460GT only
29	E3TxFH	Parity Error EMAC3 Tx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC3 Tx FIFO	PPC460GT only
30	E3RxFI	Parity Error EMAC3 Rx FIFO Bits 0:63 0 No forced parity error 1 Force parity error into EMAC3 Rx FIFO	PPC460GT only
31	E3RxFH	Parity Error EMAC3 Rx FIFO Bits 64:127 0 No forced parity error 1 Force parity error into EMAC3 Rx FIFO	PPC460GT only

14.6 Initialization Software Requirements

After a reset operation occurs, the PPC460EX/EXr/GT is initialized to the minimum configuration required to enable the fetching and execution of the software initialization code. This ensures that the core will always function the same way during the execution of this code. Initialization software is required to complete the configuration of the processor core and the rest of the on-chip and off-chip system.

The system must provide non-volatile memory (or memory initialized by some means other than the PPC460EX/EXr/GT) at the real address corresponding to effective address 0xFFFFFFFC, and throughout the rest of the initial program memory page. The instruction at the initial address must be an unconditional branch back to the beginning of the initialization software sequence.

The initialization software functions described in this section perform the configuration tasks required to prepare the PPC460EX/EXr/GT to boot an operating system and subsequently execute an application program.

The initialization software must also perform functions associated with hardware resources that are outside the PPC460EX/EXr/GT. This section makes reference to some of these functions, but their full scope is discussed in the chapters describing peripheral operations.

Initialization software must perform the following tasks in order to fully configure the PPC460EX/EXr/GT. For more information about the various functions referenced in the initialization sequence, see the corresponding chapters of this document.

1. Branch back from effective address 0xFFFFFFFC to the start of the initialization sequence.
2. Invalidate the instruction cache (**iccci**).
3. Invalidate the data cache (**dccci**).
4. Synchronize memory accesses (**msync**).

   This step forces any data PLB operations that may have been in progress before the reset operation to complete, thus allowing subsequent data accesses to be initiated and completed properly.
5. Clear DBCR0 register (disable all debug events).

   Although the PPC460EX/EXr/GT is designed to reset some of the debug event enables during the reset operation (as specified in *Table 14-1* on page 287), this is not required by the architecture and the initialization software must not assume this behavior. Software must disable all debug events in order to prevent unpredictable behavior on the trace interface to the core.
6. Clear DBSR register (initialize all debug event status).

   Although the PPC460EX/EXr/GT is defined to reset the DBSR debug event status bits during the reset operation (as specified in *Table 14-1* on page 287), this is not required by the architecture and the initialization software must not assume this behavior. Software must clear all such status in order to prevent unpredictable behavior on the JTAG interface to the core.
7. Initialize CCR0 register.
   1. Enable/disable broadcast of instructions to auxiliary processor (save power if no AP attached).
   2. Enable/disable broadcast of trace information (save power if not tracing).
   3. Enable/configure or disable speculative instruction cache line prefetching.
   4. Specify behavior for **icbt** and **dcbt**/**dcbtst** instructions.
   5. Enable/disable gathering of separate store accesses.
   6. Enable/disable hardware support for misaligned data accesses.
   7. Enable/disable parity error recoverability (recoverability lowers load/store performance marginally).

8. Enable/disable cache read of parity bits, depending on software compatibility requirements.

8. Initialize CCR1 register.
    1. Enable/disable full-line flushes as needed.
    2. Disable force cache-op miss (FCOM) and various parity error insertion (xxxPEI).
    3. Users may want to initialize CCR1[TCS] here, or in the timer facilities section.
9. Configure instruction and data cache regions.

    These steps must be performed before enabling the caches, by setting the caching inhibited storage attribute of the corresponding TLB entry to 0.
    1. Clear the instruction and data cache normal victim index registers (INV0–INV3, DNV0–DNV3).
    2. Clear the instruction and data cache transient victim index registers (ITV0–ITV3, DTV0–DTV3).
    3. Set the instruction and data cache victim limit registers (IVLIM and DVLIM) according to the desired size of the normal, locked, and transient regions of each cache.
10. Setup TLB entry for the initial program memory page.

    The PPC460EX/EXr/GT only initializes an architecturally-invisible shadow TLB entry during the reset operation, and since all shadow TLB entries are invalidated upon any context synchronization This means that special care must be taken during the initialization sequence to prevent any such context-synchronizing operations (such as interrupts and the **isync** instruction) until after this step is completed and an architected TLB entry has been established in the TLB. Particular care should be taken to avoid store operations, because write permission is disabled upon reset; an attempt to execute any store operation would result in a data storage interrupt, thus invalidating the shadow TLB entry.
    1. Initialize MMUCR
        - Specify TID field to be written to TLB entries.
        - Specify TS field to be used for TLB searches.
        - Specify store miss allocation behavior.
        - Enable/disable transient cache mechanism.
        - Enable/disable cache locking exceptions.
    2. Write TLB entry for initial program memory page.
        - Specify EPN, RPN, ERPN, and SIZE as appropriate for system.
        - Set valid bit.
        - Specify TID = 0 (disable comparison to PID) or else set PID register to matching value.
        - Specify TS = 0 (system address space) or else set MSR[IS,DS] to correspond to TS = 1.
        - Specify storage attributes (W, I, M, G, E, U0–U3) as appropriate for system.
        - Enable supervisor mode fetch, read, and write access (SX, SR, SW).
    3. Initialize PID register to match TID field of TLB entry (unless using TID = 0).
    4. Setup for subsequent MSR[IS,DS] initialization to correspond to TS field of TLB entry.

        Only necessary if the TS field of the TLB entry is being set to 1 (MSR[IS,DS] already reset to 0).
        - Write the new MSR value into SRR1.
        - Write the address from which to continue execution into SRR0.
    5. Set up for subsequent change in instruction fetch address.

Only necessary if EPN field of TLB entry changed from the initial value ($EPN_{0:19} \neq$ 0xFFFFF).

- Write the initial/new MSR value into SRR1.
- Write the address from which to continue execution into SRR0.

6. Fully initialize the TLB (tlbwe to all three words of each TLB entry; tlbre to TLB entries that are not fully initialized may result in parity exceptions).
7. Context synchronize to invalidate shadow TLB contents and cause new TLB contents to take effect.
   - Use **isync** if not changing MSR contents and not changing the effective address of the rest of the initialization sequence.
   - Use **rfi** if changing MSR to match new TS field of TLB entry (SRR1 will be copied into MSR, and program execution will resume at value in SRR0).
   - Use **rfi** if changing next instruction fetch address to correspond to new EPN field of TLB entry (SRR1 will be copied into MSR, and program execution will resume at value in SRR0).

   Instruction and data caches will now start being used if the corresponding TLB entry has been set up with the caching inhibited storage attribute set to 0. Initialization software can now branch outside of the initial 4KB memory region as controlled by the address and size of the new TLB entry and/or any other TLB entries which have been set up.

11. Initialize interrupt resources.
    1. Initialize IVPR to specify high-order address of the interrupt handling routines.

       Ensure that the corresponding address region is covered by a TLB entry (or entries).
    2. Initialize IVOR0–IVOR15 registers (individual interrupt vector addresses).

       Ensure that the corresponding addresses are covered by a TLB entry (or entries).

       Because the low order four bits of IVOR0–IVOR15 are reserved, the values written to those bits are ignored when the registers are written, and are read as zero when the registers are used. Therefore, all interrupt vector offsets are implicitly aligned on quadword boundaries. Software must ensure that all interrupt handlers are quadword aligned.
    3. Setup corresponding memory contents with the interrupt handling routines
    4. Synchronize any program memory changes as required. Refer to the section about self-modifying code in the *PPC440 Processor User's Manual* for more information about the instruction sequence necessary to synchronize changes to program memory before executing the new instructions.)
12. Configure debug facilities as desired.
    1. Write DBCR1 and DBCR2 to specify IAC and DAC event conditions.
    2. Clear DBSR to initialize IAC auto-toggle status.
    3. Initialize IAC1–IAC4, DAC1–DAC2, DVC1–DVC2 registers to needed values.
    4. Write MSR[DWE] to enable Debug Wait mode (if desired).
    5. Write DBCR0 to enable desired debug mode(s) and event(s).
    6. Context synchronize to establish new debug facility context (**isync**).
13. Configure timer facilities as needed.
    1. Write DEC to 0 to prevent decrementer exception after TSR is cleared.
    2. Write TBL to 0 to prevent Fixed Interval Timer and Watchdog Timer exceptions after TSR is cleared, and to prevent increment into TBH prior to full initialization.
    3. CCR1[TCS] (Timer Clock Select) can be initialized at this time, or earlier with the rest of the CCR1.

4. Clear TSR to clear all timer exception status.
5. Write TCR to configure and enable timers as desired.

   Software must be careful with respect to the enabling of the Watchdog Timer reset function, because once this function is enabled, it cannot be disabled except by reset itself.
6. Initialize TBH value as desired.
7. Initialize TBL value as desired.
8. Initialize DECAR to needed value (if enabling the auto-reload function).
9. Initialize DEC to needed value.

14. Initialize facilities outside the processor core that might potentially cause asynchronous interrupt requests (including DCRs and/or other memory-mapped resources).

    This must be done prior to enabling asynchronous interrupts in the MSR.
15. Initialize the MSR to enable interrupts as needed.
    1. Set MSR[CE] to enable/disable Critical Input and Watchdog Timer interrupts.
    2. Set MSR[EE] to enable/disable External Input, Decrementer, and Fixed Interval Timer interrupts.
    3. Set MSR[DE] to enable/disable Debug interrupts.
    4. Set MSR[ME] to enable/disable Machine Check interrupts.

       Software should first check the status of the ESR[MCI] field and MCSR[MCS] field to determine whether any Machine Check exceptions have occurred after these fields were cleared by reset and before Machine Check interrupts were enabled (by this step). Any such exceptions would have set ESR[MCI] or MCSR[MCS] to 1, and this status can only be cleared explicitly by software. After the MCSR[MCS] field is known to be clear, software must clear the MCSR status bits (MCSR[1:8]) in order to avoid possible confusion upon later service of a Machine Check interrupt. Once MSR[ME] has been set to 1, subsequent Machine Check exceptions will result in a Machine Check interrupt.
    5. Context synchronize to establish new MSR context (**isync**).
16. Initialize any other processor core resources as required by the system (GPRs, SPRGs, and so on).
17. Initialize any other facilities outside the processor core as required by the system.
18. Initialize system memory as required by the system software.

    Synchronize any program memory changes as required. (Refer to the section about self-modifying code in the *PPC440H6 Processor User's Manual* for more information about the instruction sequence needed to synchronize changes to program memory before executing the new instructions.).
19. Start the system software.

    System software is generally responsible for initializing and/or managing the remaining MSR fields, including:
    1. MSR[FP] to enable or disable the execution of floating-point instructions.
    2. MSR[FE0,FE1] to enable/disable Floating-Point Enabled exception type Program interrupts.
    3. MSR[PR] to specify user mode or supervisor mode.
    4. MSR[IS,DS] to specify application address space or system address space for instructions and data.
    5. MSR[WE] to place the processor into Wait State (halt execution pending an interrupt).

15. Timer Facilities

The PPC460EX/EXr/GT processor provides four timer facilities: a time base, a Programmable Interval Timer (PIT), a fixed interval timer (FIT), and a watchdog timer. The PIT is a Special Purpose Register (SPR). These facilities, which are driven by the same base clock, can, among other things, be used for:

- Time-of-day functions
- Data logging functions
- Peripherals requiring periodic service
- Periodic task switching

Additionally, the watchdog timer can help a system to recover from faulty hardware or software.

These timer facilities are part of the PPC440 processor. Refer to the *PPC440H6 Processor User's Manual* for details.

16. Debugging

The debug facilities of the PPC460EX/EXr/GT include support for debug modes for debugging during hardware and software development, and debug events that allow developers to control the debug process. Debug registers control the debug modes and debug events. The debug registers are accessed through software running on the processor or through a JTAG debug port. The debug interface is the JTAG debug port. The JTAG debug port can also be used for board test.

The debug modes, events, controls, and interface provide a powerful combination of debug facilities for a wide range of hardware and software development tools.

The JTAG interface is a part of the PPC440 processor. Refer to the *PPC440H6 Processor User's Manual* for details.

17. Clock and Power Management

The PPC460EX/EXr/GT provides a clock and power management (CPM) controller that reduces power dissipation by stopping clocks in unused or dormant functional units. Use of the CPM controller requires careful programming and special consideration to avoid compromising system and functional unit integrity.

17.1 Overview

The CPM controller supports three different types of sleep interfaces to the functional units:

- In a CPM class 1 interface, the CPM_Sleep_N signal is asserted by the CPM controller when a register bit is set by software. The functional unit is unconditionally put to sleep. There is no other communication with the functional unit.
- In a CPM class 2 interface, the functional unit uses a combination of its internal state and external inputs to determine whether or not it can be put to sleep. If sleeping is permitted, the functional unit asserts the Sleep_Req signal to the CPM controller that responds by asserting CPM_Sleep_N if the enable for that unit is set. The CPM_Sleep_N signal to a class 2 unit is deasserted when the CPM controller enable bit for that unit is reset, or when the unit deasserts its Sleep_Req signal.
- The CPM class 3 interface has a CPM_SleepInit signal that is asserted by the CPM controller to request that a functional unit go to sleep. If the unit can sleep, it asserts the Sleep_Req signal to the CPM controller. The CPM_Sleep_N signal is then asserted by the CPM controller to shut off the class 3 clocks in the functional unit. The functional unit or the CPM controller can end the sleep state. If the CPM controller enable bit for the unit is reset, the CPM controller immediately deasserts CPM_SleepInit and CPM_Sleep_N.

17.2 CPM Registers

Table 17-1 lists the registers used to program the CPM controller.

Table 17-1. CPM Registers

Mnemonic	Register	DCR Address	Access	Page
CPM0_ER	CPM0 Enable Register	0x0160	Read/Write	309
CPM0_FR	CPM0 Force Register	0x0161	Read/Write	311
CPM0_SR	CPM0 Status Register	0x0162	Read Only	311

17.2.1 CPM Enable Registers (CPM0_ER)

The CPM0_ER bits enable the process of putting a functional unit to sleep. The class of a unit determines how its interface signals are controlled when the bit associated with the unit is set to 1.

Class 1 — When an associated CPM0_ER bit is set to 1, the CPM_Sleep_N signal to the class 1 unit is asserted. When the bit is set to 0, CPM_Sleep_N is deasserted.

Class 2 — When an associated CPM0_ER bit is set to 1, and the Sleep_Req signal from the class 2 unit is asserted (the unit is requesting sleep state), CPM_Sleep_N to the class 2 unit is asserted. When the bit is set to 0, the CPM_Sleep_N signal is deasserted.

Class 3 — When an associated CPM0_ER bit is set to 1, the CPM_SleepInit signal to the class 3 unit is asserted (the CPM controller is requesting permission to put the unit to sleep). When the class 3 unit activating the Sleep_Req in response, (the unit is giving permission to be put to sleep), CPM_Sleep_N signal to the class 3 unit is asserted. When the bit is set to 0, CPM_SleepInit and CPM_Sleep_N are deasserted.

Figure 17-1. CPM0 Enable Register (CPM0_ER)

0	IIC0	Inter-Integrated Circuit 0	Class 3
1	IIC1	Inter-Integrated Circuit 1	Class 3
2	EIP94	Security Engine	Class 1
3	EIPPKP	PKA and TRNG	Class 1
4	SRAM	Internal SRAM Controller	Class 2
5	FPU	PPC440 Floating Point Unit	Class 2
6	CPU	PPC440 Processor	Class 2
7	OCM	On-Chip Memory Controller	Class 2
8	BGO	PLB to OPB Bridge	Class 2
9	NDFC	Nand Flash Controller	Class 2
10	EBC	External Bus Controller	Class 2
11	DMA2P40	DMA (DMA2P40)	Class 2
12	HSDMA	I2O/DMA	Class 1
13	PCIE0	PCI Express 0 (x1)	Class 1
14	PCIE1	PCI Express 1 (x4)	Class 1
15	SRIO	Serial RapidIO	PPC460GT only Class 1
16	PLB4XAHB	PLB to AHB Bridge, AHB Arbiter	PPC460EX only Class 1
17	USBOTG	USB2.0 OTG Controller	PPC460EX only Class 1
18	USBHOST	USB2.0 Host Controller	PPC460EX only Class 1
19	SATA	Serial ATA, AHB DMA Controller, AHB Inter-Connect Matrix	PPC460EX only Class 1
20	GPIO	General Purpose IO	Class 1
21	GPT	General Purpose Timer	Class 1
22	UART01	Universal Asynchronous Receiver/Transmitter 0 and 1	Class 1
23	UART23	Universal Asynchronous Receiver/Transmitter 2 and 3	Class 1

24	TMRCLK	CPU Timer	Class 1
25	MAL	Memory Access Layer for Ethernet	Class 1
26:27	EMAC0:1	Ethernet Media Access Controller 0:1	Class 1
28:29	EMAC2:3	Ethernet Media Access Controller 2:3	PPC460GT only Class 1
30:31	TAHOE0:1	TCP/IP Acceleration on Ethernet 0:1	Class 1

17.2.2 CPM Force Register (CPM0_FR)

Setting a CPM0_FR bit forces assertion of the $\overline{\text{CPM\_Sleep}}$ signal to the functional unit. For a class 1 unit, this is equivalent to setting the CPM0_ER bit associated with the unit. For class 2 or class 3 units, $\overline{\text{CPM\_Sleep}}$ is asserted regardless of the state of the Sleep_Req signal coming from the unit.

Figure 17-2. CPM0 Force Register (CPM0_FR)			
0:31	Same as CPR0_ER	Same as CPR0_ER.	Same as CPR0_ER.

17.2.3 CPM Status Register (CPM0_SR)

The read-only CPM0_SR shows the assertion level of all sleep signals.

Figure 17-3. CPM Status Register (CPM0_SR)			
0:31	Same as CPR0_ER	Same as CPR0_ER.	Same as CPR0_ER.

Part IV. External Interfaces

18. Security Function

The built-in security feature is a cryptographic engine attached to the 128-bit PLB with built-in DMA and interrupt controllers.

Features include:

- Federal Information Processing Standard (FIPS) 140-2 design
- Support for an unlimited number of Security Associations (SA)
- Different SA formats for each supported protocol (IPsec/SSL/TLS/sRTP)
- Internet Protocol Security (IPsec) features
  - Full packet transforms, Encapsulated Security Payload (ESP) and Authentication Header (AH)
  - Complete header and trailer processing (IPv4 and IPv6)
  - Multi-mode automatic padding
  - "Mutable bit" handler for AH, including IPv4 option and IPv6 extension headers
  - Extended Sequence Number (ESN) processing for ESP and AH
- Secure Socket Layer (SSL) and Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) features
  - Packet transforms
  - One-pass hash-then-encrypt for SSL and TLS packet transforms for inbound packet using Stream Cipher
- Secure Real-Time Protocol (sRTP) features
  - Packet transforms
  - Roll Over Counter (ROC) removal and TAG insertion
  - Variable bypass offset of header length per packet
- Media Access Control Security (MACSec) features
  - MSDU (User data) encryption 0, 30, or 50 bytes offset
  - Header insertion and removal
  - SecTAG header with or without Secure Channel Identifier (SCI) field
  - 128-bit key, 96-bit IV (nonce) and 128-bit ICV
  - IV from SA record or from input buffer (as part of SecTAG)
  - ICV generation and validation
- SGT features
  - GCM-AES with 128-bit key, 96-bit IV (nonce) and 128-bit ICV
  - SecTAG header with or without Secure Channel Identifier (SCI) field
  - Replay protection "Strict order Mode" and "Out of Order Mode"
  - Header insertion and removal
  - ICV generation and validation
- IPsec/SSL security acceleration function
- DES, 3DES, AES, ARC-4 encryption (no support for hashing of zero length messages)
- MD-5, SHA-1, and SHA-2 (224-, 256-, 384-, and 512-bit) hashing, HMAC encrypt-hash and hash-decrypt
- KASUMI algorithm
- Public key acceleration (PKA) for RSA, DSA and Diffie-Hellman
- True (TRNG) or pseudo (PRNG) random number generators
  - Non-deterministic true random numbers
  - Pseudo random numbers with lengths of 8 or 16 bytes
  - ANSI X9.17 Annex C compliant using a DES algorithm
- Interrupt controller
  - Fifteen programmable, maskable interrupts
  - Initiate commands by means of an input interrupt
  - Sixteen programmable interrupts indicating completion of certain operations
  - All interrupts mapped to one level- or edge-sensitive programmable interrupt output
- DMA controller
  - Autonomous, 4-channel
  - 1024 words (32 bits/word) per DMA transfer
  - Scatter/gather capability with byte aligned addressing

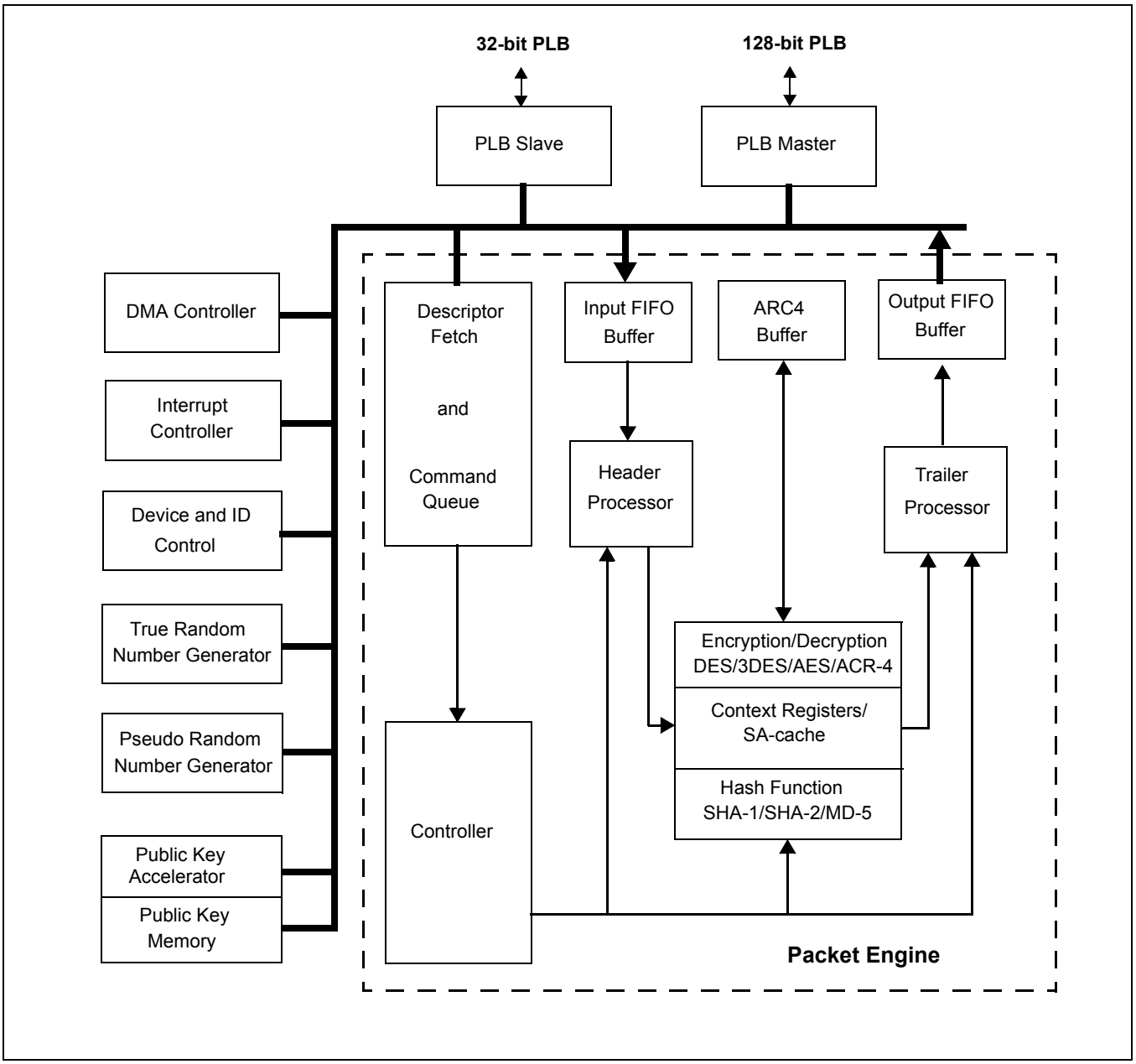
18.1 Functional Description

The following sections provide a brief functional description of the operation of the security function.

18.1.1 Block Diagram

*Figure 18-1*shows the functional block diagram of the security function within the PPC460EX/EXr/GT.

Figure 18-1. Security Function Block Diagram



18.1.2 PLB Interface

The PLB Slave interface allows the processor to configure the security function and to access functions such as the true random number generator and public key accelerator. The PLB Master interface allows the security function to fetch descriptors and cryptographic context information, and transfer packet data to and from the Packet Engine. The system PLB clock is the source for the main security function clock. The frequency range for the PLB clock is defined in the in the respective 460EX, 460EXr, or 460GT Data Sheet.

18.1.3 Packet Engine

The Packet Engine is the main feature in the security function. It is optimized to off-load time-consuming cryptographic operations from the processor and contains functions for both symmetric encryption and hashing. The Packet Engine runs autonomously, performing DMA transfers of control information and payload data to and from the main bus. The DMA process incorporates flow-control to guarantee proper data flow. The Packet Engine allows parallel and pipe lined encryption and hashing operations, significantly reducing the latency and processing time for packets that need multiple operations applied.

18.1.3.1 Packet Engine Architecture

The Packet Engine provides hardware acceleration for IPSec, SSL/TLS, DTLS, SRTP, MACSec, and SGT L2/L3 security protocols. The engine supports most cipher suites used by these protocols for data encryption, decryption and authentication. For all protocols, except SRTP, it provides header processing, sequence number checking, payload encryption and decryption, packet authentication and protocol padding generation and verification as required by the protocol standard.

The Packet Engine allows parallel and pipelined encryption and hashing operations, to significantly reduce the latency and process time for packets that need multiple operations applied.

The Packet Engine subsystem contains the algorithm engines, surrounded by intelligent packet pre- and postprocessors that are responsible for inserting and removing the necessary fields in the packet header and trailer. These include:

- Inserting Protocol Header (SPI, Sequence Number, version number, Next Header)
- Inserting Protocol Trailer (Padding data, Pad Length, Next Header)
- Updating IP Header fields (Next Header, Length, Checksum)
- Mutable bit processing and ICV insertion for AH
- Inserting IV (Initialization Vector)

In addition, the pre-, post- processors are responsible for:

- Validating Sequence Numbers
- Validation Integer Check Value (ICV) or Message Authentication Code (MAC) or authentication TAG
- Validating Pad Values and Pad Length
- Stripping Headers and Trailers
- Stripping IV
- Stripping Pad

The algorithm engines use separate input/output buffers, which transfer data between the I/O buffers and the cryptographic and hash engine completely autonomously.

Data for the Packet Engine is transferred between the input/output buffers by means of the security feature's DMA function without requiring processor intervention. The packet engine handles all internal data movements between I/O buffers and internal processors.

18.1.3.2 Input/Output Buffer

The data for the Packet Engine is buffered at both input and output. These buffers decouple the DMA I/O process from the encryption and hash modules inside the Packet Engine. This enables large DMA burst sizes and allows the cryptographic function to keep busy during I/O latency periods. Data is automatically moved from the input buffer through the encryption and hash functions and into the output buffer. If the output buffer is full, the process will pause until data has been read and there is space available in the output buffer.

18.1.3.3 Controller

This is the manager of the Packet Engine that allows it to autonomously process packets from either a Descriptor Ring or a direct command register set. The security function can be configured in one of three command modes.

The first is the Autonomous Ring mode, in which case the security function polls a descriptor ring in memory space. Various polling parameters may be configured at initialization. An input interrupt allows the processor to issue an interrupt to the security function in order to trigger a descriptor fetch.

The second mode is the Target Command mode. A packet command register set within the security function is written by the processor to initiate a packet operation. This eliminates much of the I/O bus overhead of polling for descriptors, but requires the processor to synchronously initiate packet processing.

The third mode is the Direct Host DMA mode. In this mode the processor must provide the packet command register set, the SA and the packet data. This means there are no autonomous read or write transfers; the processor always initiates the transactions.

When a packet operation is complete, the security function provides a result structure that indicates the status and provides the new length, pad result, IPsec next header field, and so on. The security function can write this result to a ring in memory using the PLB Master interface, or the processor can read the result structure from internal security function registers using the PLB Slave interface. An optional interrupt can be generated by the security function at completion of packet processing.

18.1.3.4 Header and Trailer Processor

This header and trailer processors implement all of the protocol header and trailer processing for the security protocols. This includes padding and optional insertion of an IV at the beginning of a packet. When implementing IPsec operations, these functions performs all IPsec header and trailer insertion and removal for both the ESP and AH as shown in the *Table 18-1* and *Table 18-2*:

Table 18-1. IPsec ESP Header Processing

Element	Size	Outbound	Inbound
SPI (Security Parameters Index)	32 bits	Insert.	Extract, verify against SA-record.
Replay Counter/Sequence Number	32 bits	Increment and insert.	Extract, verify against expected count and 64-bit window mask; update count and mask after authentication passes.
IV (Initialization Vector)	variable	Insert random value.	Extract and load into cryptographic function.
Padding	0–255B	Insert padding up to 255B.	Strip padding (selectable).
Next header	8 bits	Insert into pad trailer field.	Extract and report in result descriptor.
ICV (Integrity Check Value)	96 bits	Calculate and insert.	Extract and verify. Optionally discard.
Note: Supports RFC-2024.			

Table 18-2. IPsec AH Processing

Element	Size	Outbound	Inbound
Outer IP header	20–60 B	Update length, next header, and header checksum in IP header. Zero the "mutable bit" fields as HMAC is calculated.	Parse the outer IP header and options (IPv4) or extension headers (IPv6) to locate AH header. Zero the "mutable bit" fields as HMAC is calculated.
SPI (Security Parameters Index)	32 bits	Insert.	Extract, verify against SA-record.
Replay Counter/Sequence Number	32 bits	Increment and insert.	Extract, verify against expected count and 64-bit window mask.
Next header	8 bits	Insert into AH header field.	Extract and report in result descriptor.
ICV (Integrity Check Value)	96 bits	Calculate and Insert.	Extract and verify. Optionally discard.
Note: Supports RFC-2024.			

18.1.3.5 Encryption/Decryption

Performs high-speed encryption/decryption operations, including multiple block cipher and stream cipher algorithms.

Block ciphers: AES, DES and Triple DES. All four standard modes are supported:
- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- 64-bit Output FeedBack (OFB) for DES/Triple-DES
- 1-bit, 8-bit and 64-bit Cipher FeedBack (CFB) for DES/Triple-DES and 128-bit CFB for AES.

Stream ciphers: AES Counter and ARC4. Two AES counter modes and two ARC4 modes are supported:
- AES Counter Mode (CTR) as defined for IPsec ESP.
- AES Integer Counter Mode (ICM) as defined for SRTP.

- ARC4 "Stateless" and "Stateful" mode

All AES modes are supported with key lengths of 128, 192, and 256 bits except for AES-GCM which is supported with the key length of 128 bits.

The high performance encryption/decryption implementations are highly pipelined and execute multiple DES rounds per clock cycle and a full AES round in a single clock cycle, minimizing packet latency. Key scheduling is automatic and performed in parallel with the encryption/decryption operation.

High performance ARC4 encryption/decryption using 256-byte S-box and 256-byte shadow S-box allows up to 2 permutations simultaneously. ARC4 key lengths up to 128 bits are supported.

Hardware support for padding insertion, verification and removal further accelerates the encryption/decryption process.

18.1.3.6 Hash Function

Tightly coupled with the encryption/decryption function, the hash function provides hardware accelerated one-way hashing operations. The security function supports the MD-5, SHA-1, and SHA-2 algorithms. An HMAC processing unit is included as well, which automatically handles the "outer" hash calculations, and GMAC, AES-XCBC-MAC and GHASH functions.

The hash function contains an intelligent “mutable bit” handler in hardware, which adds a significant performance boost to IPsec AH processing. The security function automatically clears the necessary fields in an IPv4 or IPv6 header as well as the appropriate IPv4 Options fields or IPv6 Extension Header fields.

- The Message Authentication Code (MAC) is used for SSL Operations.
- The Hashed MAC (HMAC) is used for Basic, IPsec, TLS and SRTP Operations.

18.1.3.7 Command Queue and Context Registers

The control data that specifies the processing for each packet is provided in two elements: The Packet Descriptor and the Security Association (SA) record.

The real-time operation of the Packet Engine is controlled by packet commands that are written into a command queue that drives the Packet Engine. The command queue can be written by the processor, or filled with descriptor data that is automatically fetched from a descriptor ring in memory by the Packet Engine.

The SA-record is a packed structure that contains the remainder of the information needed by the Packet Engine to process a packet. The data is stored in context registers that can be written by the processor, or filled with SA-record data that is automatically fetched from memory by the Packet Engine.

18.1.3.8 Using the Packet Engine

This section will discuss some general principles and methods for using the Packet Engine efficiently.

Selection of Descriptor Modes

The first decision to be made relates to the Packet Engine's modes of operation.

The major choice is whether the processor must supply the packet descriptor, SA-record data and packet data, or whether data is fetched autonomously by the Packet Engine.

The Direct Host DMA mode allows the processor to have full control over the Packet Engine through the PLB Slave Interface. However, this mode can be very inefficient since much communication is required between the processor and the security processor.

For the PLB Master mode, there is a choice among enabling Descriptor Ring processing, Autonomous Ring mode, or whether the application will target-write commands (Target Command mode) directly into the on-chip Command Queue. This Command Queue appears as a set of 5-word packet-command registers.

In general, the Autonomous Ring mode is more efficient because it allows the security processor and the processor to operate asynchronously, and allows queuing of multiple packets to be processed. This minimizes starving of the security processor and provides the highest possible throughput.

If Descriptor Ring mode is enabled, then another mode choice is related to the grouping of the Packet Descriptors, the Packet Data, and the SA structure. Each of these elements can be in distinct locations, with the Descriptor pointing to both Packet Data and SA location. Or, the data elements may be linked together so that one piece follows another.

Another choice if Descriptor Ring mode is enabled, is whether the processor will write Packet Descriptors directly into the on-chip Command Queue (one entry) or whether the security processor should fetch them from a External Packet Descriptor Ring (PDR) in external memory (Descriptor Fetch Engine enabled). This choice is made with the PE Ring Size register (0 specifies that the processor writes directly to the internal Command Queue).

Assuming the External PDR mode is used, a decision must be made as to how the security processor is made aware of new descriptors appearing on the PDR. The two choices are Polling or Interrupt.

In the Polling configuration, the security processor simply polls the PDR until it finds one or more entries that have the ownership bits set to the security processor. The frequency of polling, and therefore the amount of bus bandwidth that is consumed, is configurable in the PE Ring Poll register. Separate controls are provided for normal polling and for poll re-tries when the descriptor read is not ready.

In the Interrupt configuration, the Host populates one or more Packet Descriptors and then issues an interrupt to the security processor to tell it to fetch the descriptor(s) and begin processing. This mode usually imposes less bus overhead on the system. It also offers controlled processing latency, since the processor specifies when descriptors will be processed.

Result Descriptors (Autonomous Ring mode only)

When the security processor finishes processing a packet, it writes a Result Descriptor to a Result Descriptor Ring (RDR). The RDR can be thought of as a mirror to the PDR. The user specifies on what bus and at what base address the RDR ring exists. This flexibility is provided so that the results can be efficiently posted to the processor without requiring the processor to perform Master reads.

Generally, if the processor does not write packet descriptors to the internal Command Queue, then the PDR and RDR should be overlaid on top of each other. This minimizes the memory consumed for descriptors and reduces the memory bus utilization. If the PDR and RDR are in separate external locations, then an additional update is also required to the ownership bits in the PDR to prevent the security processor from re-processing old descriptors. (This additional update may be disabled in the CRYP0_PE_DMA_CF register if the processor wishes to prevent ring wrapping.)

SA-Record Storage

In Autonomous Ring mode and Target Command mode, the SA-records are always stored in a memory area that the security processor can access by means of reads across the PLB Master. Each SA-record is 128-bytes in size, and there is no limit to how many SA's the SECURITY ENGINE can support.

In the Direct Host DMA mode, the security processor cannot perform a master fetch of the SA-record data, so it must be presented by the processor after writing the descriptor command and before writing packet data to the input buffer.

The SA registers are generally accessed by the processor only in Direct Host DMA Mode. Although they contain meaningful values in Autonomous Ring mode and Target Command mode, it is recommended that these registers be used only for debugging purposes in these modes.

18.1.4 DMA Controller

The security function incorporates a high-performance 4-channel 32-bit DMA controller which can be used to efficiently move data between memory, the Packet Engine, and the other internal security function modules. The four channels are: descriptor fetch, data I/O, scatter/gather descriptor fetch, and user-programmable.

The user programmable channel is provided for the user to initiate PLB Master DMA transfers to internal locations.

18.1.5 Public Key Accelerator and Public Key Memory

The Public Key Accelerator (PKA) provides high-performance large-vector arithmetic functions, such as multiply, add, subtract, or exponentiation. Vector sizes up to 2048 bits are supported. The Public Key Memory (PKM) stores the intermediate results during large number arithmetic operations and holds the final result.

Basic Operations:
- Addition, subtraction and combined addition/subtraction
- Shift right or left
- Multiplication, division (with and without quotient)
- Compare and copy

Complex Operations:
- Unsigned value modular exponentiation
- Signed value modular exponentiation (using Chinese Remainders Theorem (CRT) method)
- Modular inversion
- ECC point addition/doubling on elliptic curve
- ECC point multiplication on elliptic curve

18.1.5.1 Operation

The Public Key Co-Processor (PKCP) operates directly on the Public Key Memory (PKM). Operand and result vectors are stored in the PKM. The vectors are sequentially cycled through the processing engines of the PKCP and intermediate products from large or complex operations are temporarily stored in this PKM.

The processor is responsible for configuring the PKA for the intended operation, providing proper operand data, and allocating space for the result vector. The processor can access the PKCP through the PLB Slave interface to program the PKM vector locations, vector lengths and the function to be performed. It can also read the status and result information. The processor can access the PKM through the PLB Slave interface to write operands and read result vectors.

Once the processor has initiated an operation, the PKCP must have unrestricted access to the PKM RAM.

18.1.5.2 Functional Description

The PKA is configured for an operation through a set of registers accessed from the PLB. Using these registers, the processor specifies the function, along with the length and location of the operand and result vectors. Both operand and result vectors are stored in the PKM external to the PKCP. The PKM resides on the internal bus, allowing the processor to write it with the input operands, and read the result vectors.

The PKA engine accesses these vectors through the memory interface. During computation, the vectors are sequentially cycled through the math processors of the PKCP. Intermediate results from large or complex operations are routinely written back to this memory for temporary storage. Memory allocation must be performed with consideration to its secondary role as the working space for the PKA.

When the processor sets CRYP0_PKA_FUNC[RUN], the PKA engine begins an operation. This bit remains set until the PKA engine has completed the operation and then is cleared. The processor can either poll this register or enable the PKA interrupt in the security processor Interrupt Controller to determine when an operation is complete.

The PKA Engine with CRT support provides the following basic operations:
- Large vector addition, subtraction and combined addition/subtraction
- Large vector shift right or left
- Large vector multiplication, division (with and without quotient)
- Large vector compare and copy

The PKA Engine with CRT support provides the following complex operations:
- Large vector unsigned value modular exponentiation
- Large vector unsigned value modular exponentiation using the 'Chinese Remainders Theorem' ('CRT') method with pre-calculated Q inverse vector

These operations are done under control of the Sequencer. Depending on the content of the Program ROM, the Sequencer implements modular exponentiations using only the PKCP. In both cases, the Sequencer hides the fact that the actual exponentiation is done using numbers in the Montgomery domain. For improved performance, the Sequencer also employs exponent recoding techniques that use a table with pre-calculated odd powers.

18.1.5.3 PKA RAM

The vectors (big numbers) that are the input and output of the PKA operations are stored in the PKA RAM. Each vector consists of a sequence of (32-bit) words, stored 'little endian' in a contiguous block of memory with the least significant word at the lowest address of that memory block and bit [0] of the vector in bit [0] of that word.

Note: All input and output vectors must start at an even 32-bit word address, i.e. be aligned to an 8-byte boundary in PKA RAM.

18.1.5.4 PKCP Operations

Table 18-3 lists the arguments and results for each PKCP operation.

Table 18-3. Summary of PKCP Vector Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
Multiply	A x B → C	Multiplicand	Multiplier	Product	N/A
Add	A + B → C	Addend	Addend	Sum	N/A
Subtract	A – B → C	Minuend	Subtrahend	Difference	N/A
AddSub	A + C – B → D	Addend	Subtrahend	Addend	Result
Right Shift	A >> Shift → C	Input	N/A	Result	N/A
Left Shift	A << Shift → C	Input	N/A	Result	N/A
Divide	A mod B → C, A div B → D	Dividend	Divisor	Remainder	Quotient
Modulo	A mod B → C	Dividend	Divisor	Remainder	N/A
Compare	A = B, A < B, A > B	Input1	Input2	N/A	N/A
Copy	A → C	Input	N/A	Result	N/A

To obtain correct result, the input vectors must meet the requirements presented in *Table 18-4*. Note that:
- Input restrictions are not checked by the PKCP
- A_Len and B_Len indicate the size of vectors A and B in (32-bit) words
- Max_Len equals 64 (32-bit) words, i.e. the standard maximum vector size is 2048 bit

Table 18-4. Restrictions on Input Vectors for PKCP Operations

Function	Requirements
Multiply	1) 0 < A_Len, B_Len ≤ Max_Len
Add	1) 0 < A_Len, B_Len ≤ Max_Len
Subtract	1) 0 < A_Len, B_Len ≤ Max_Len 2) Result must be positive (A ≥ B)
AddSub	1) 0 < A_Len ≤ Max_Len (B and C operands have A_Len as length, B_Len ignored) 2) Result must be positive ((A + C) ≥ B)
Right Shift	1) 0 < A_Len ≤ Max_Len
Left Shift	1) 0 < A_Len ≤ Max_Len
Divide, Modulo	1) 1 < B_Len ≤ A_Len ≤ Max_Len 2) Most significant 32-bit word of B operand cannot be zero
Compare	1) 0 < A_Len ≤ Max_Len (B operand has A_Len as length, B_Len ignored)
Copy	1) 0 < A_Len ≤ Max_Len

The CPU is responsible for allocating a block of contiguous memory in PKA RAM for the result vector(s). *Table 18-5* indicates how much memory should be allocated for the result vector(s).

Table 18-5. PKCP Result Vector Memory Allocation

Function	Result Vector	Result Vector Length (in 32-bit words)
Multiply	C	A_Len + B_Len + 6 (the 6 'scratchpad' words should be discarded)
Add	C	Max(A_Len, B_Len) + 1
Subtract	C	Max(A_Len, B_Len)
AddSub	D	A_Len + 1
Right Shift	C	A_Len
Left Shift	C	A_Len + 1 (when Shift Value is non-zero) A_Len (when Shift Value is zero)
Divide	C	Remainder → B_Len + 1 (one 'scratchpad' word should be discarded)
	D	Quotient → A_Len – B_Len + 1
Modulo	C	Remainder → B_Len + 1
Compare	None	Compare updates the PKA_COMPARE register
Copy	C	A_Len

Input vectors for an operation are always allowed to overlap in memory (partially or completely). *Table 18-6* gives restrictions for the overlap of output and input vectors of the operations.

Table 18-6. PKCP Result Vector / Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
Multiply	C	No overlap with A or B vectors allowed
Add, Subtract	C	May overlap with A and/or B vector, provided the start address of the C vector does not lie above the start address of the vector(s) with which it overlaps
AddSub	D	May overlap with A, B and/or C vector, provided the start address of the D vector does not lie above the start address of the vector(s) with which it overlaps
Right Shift, Left Shift	C	May overlap with A vector, provided the start address of the C vector does not lie above the start address of the A vector
Divide	C	No overlap with A, B or D vectors allowed
	D	No overlap with A, B or C vectors allowed
Modulo	C	No overlap with A or B vectors allowed
Compare	None	Compare does not write a result vector
Copy	C	Same restrictions as for Right/Left Shift, copy of a vector to a lower address is always allowed even if source and destination overlap[a]

a. The Copy operation can be used to fill memory by breaking the overlap restrictions, but requires TWO initial (32-bit) words to be set up: To zero a block of memory, set A vector pointer to the block start, set C vector pointer two words higher and A vector length to the block length minus two (words). Fill the first two words of the block with constant zero and perform a PKCP Copy operation to zero the remainder of the block.

18.1.5.5 Sequencer Operations

The Sequencer controls modular exponentiation operations. This document assumes that the Sequencer program ROM/RAM holds code that implements the following modular exponentiation operations:

Table 18-7. Summary of ExpMod Operations

Function	Mathematical Operation	Vector A	Vector B	Vector C	Vector D
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	$C^A \bmod B \rightarrow D$	Exponent, length = A_Len	Modulus, length = B_Len	Base, length = B_Len	Result and Workspace
ExpMod-CRT	See below	Exp P followed by Exp Q at next higher even word address[a], both A_Len long	Mod P + buffer word followed by Mod Q at next higher even word address[b], both B_Len long	Q inverse, length = B_Len	Input, Result (both 2xB-Len long) and Workspace

a. If A_Len is even, Exp Q follows Exp P immediately – if A_Len is odd, there is one empty word between Exp Q and Exp P.
b. If B_Len is even, there are two empty words between Mod P and Mod Q – if B_Len is odd, there is one empty (buffer) word between Mod Q and Mod P. Note that the words following Mod P and Mod Q may be zeroed by Sequencer firmware.

The ExpMod-CRT operation performs the following computation steps[1]:

$X \leftarrow (\text{Input mod Mod P})^{\text{Exp P}} \bmod \text{Mod P}$

$Y \leftarrow (\text{Input mod Mod Q})^{\text{Exp Q}} \bmod \text{Mod Q}$

$Z \leftarrow ((((X - Y) \bmod \text{Mod P}) * \text{Q inverse}) \bmod \text{Mod P}) * \text{Mod Q}$

$\text{Result} \leftarrow Y+Z$

The ExpMod-ACT2, -ACT4 and -variable functions implement the same mathematical operation but with a differently sized table with pre-calculated 'odd powers'. The ExpMod-ACT2 function uses a table with 2 entries whereas ExpMod-ACT4 uses a table with 8 entries. The ACT4 version gives better performance but needs more memory.

ExpMod-variable and ExpMod-CRT allow a variable amount (from 1 up to and including 16) of odd powers to be selected via the register normally used to specify the number of bits to shift for shift operations.

1.These steps implement Garner's recombination algorithm after the basic exponentiations.

For a user of the PKA Engine with CRT support, the exponentiation functions appear to be extensions of the set of PKCP functions as described in *PKCP Operations* on page 324. Input and result vectors are passed just like this is done for basic PKCP operations. *Table 18-8* lists restrictions on the input and result vectors for the exponentiation operations.

Table 18-8. Restrictions on Input Vectors for ExpMod Operations

Function	Requirements
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	1) 0 < A_Len ≤ Max_Len 2) 1 < B_Len ≤ Max_Len 3) Modulus B must be odd (i.e. the least significant bit must be ONE) 4) Modulus B > 232 5) Base C < Modulus B 6) Vectors B and C must be followed by an empty 32-bit 'buffer' word
ExpMod-CRT	1) 0 < A_Len ≤ Max_Len 2) 1 < B_Len ≤ Max_Len 3) Mod P and Mod Q must be odd (i.e. the least significant bits must be ONE) 4) Mod P > Mod Q > 232 5) Mod P and Mod Q must be co-prime (their GCD must be 1) 6) 0 < Exp P < (Mod P – 1) 7) 0 < Exp Q < (Mod Q – 1) 8) (Q inverse * Mod P) 1 (modulo Mod Q) 9) Input < (Mod P * Mod Q) 10) Mod P and Mod Q must be followed by an empty 32-bit 'buffer' word

Table 18-9 shows the required scratchpad sizes for the exponentiation operations – these depend upon the PKA Engine with CRT support type . The 'M_Len' used in the table is the 'real' Modulus length (for Mod P in an ExpMod-CRT operation, for Modulus B in the other operations) in 32-bit words, i.e. without trailing zero words at the end. If the last word of the modulus vector as given is non-zero, 'M_Len' equals B_Len.

Table 18-9. ExpMod Result Vector/Scratchpad Area Memory Allocation

Function	*PKA Engine* type	Scratchpad area size (in 32-bit words) Result Vector is either M_Len or 2xM_Len 32-bit words long
ExpMod-ACT2	*PKCP*-only	5 x (M_Len + 2)
ExpMod-ACT4	*PKCP*-only	11 x (M_Len + 2)
ExpMod-variable	*PKCP*-only	(# odd powers + 3) x (M_Len + 2)
ExpMod-CRT	*PKCP*-only	(# odd powers + 3) x (M_Len + 2) + (M_Len + 2 – (M_Len MOD 2))

Note: During execution of an ExpMod-ACT2, -ACT4 or -variable operation, the last 32 bytes of the PKA RAM will be used as general scratchpad for the Sequencer's program execution. The ExpMod-CRT operation requires the last 48 bytes of the PKA RAM as scratchpad. These (fixed location) areas may not overlap with any of the input vectors and/or the D vector scratchpad area, they can be used freely when not executing an ExpMod operation.

Table 18-10. ExpMod Scratchpad Area / Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
ExpMod-ACT2 ExpMod-ACT4 ExpMod-variable	D	Scratchpad area starting at D may not overlap with any of the other vectors, except that Base C may be co-located with result vector D to save space (i.e. **PKA_CPTR = PKA_DPTR** is allowed).
ExpMod-CRT	D	Scratchpad area starting at D may not overlap with any of the other vectors, this is also the location of the main Input vector (with length 2 x B_Len)

18.1.5.6 PKA RAM Size Needed for Exponentiation Operations

For exponentiation operations, the minimum size of the PKA RAM depends on the maximum modulus length and the number of odd-powers. In addition, a fixed number of bytes are needed as scratchpad for the Sequencer firmware during execution of the exponentiation – this scratchpad must be located at the end of the PKA RAM.

Note: The Sequencer firmware allocates the 32 or 48 bytes of general scratchpad at the end of a (virtual) 8K Byte PKA RAM.

Table 18-11 indicates the required sizes of the PKA RAM to support different modulus lengths and numbers of odd powers (assuming the exponent length is the same as the modulus length). Overlapping the Base (PKA_CPTR) vector with the start of the Scratchpad/result (PKA_DPTR) area saves memory space for non-CRT operations and is therefore indicated separately:

Table 18-11. Required (PKA Engine with CRT Support) PKA RAM Sizes for Exponentiations

Modulus size (non-CRT)	**1 odd power**		**> 1 odd power (add to '1 odd power' sizes)**
	PKA_CPTR = PKA_DPTR	**PKA_CPTR PKA_DPTR**	
1024 bits	808 bytes	944 bytes	+ 136 bytes per extra odd power
2048 bits	1576 bytes	1840 bytes	+ 264 bytes per extra odd power
4096 bits	3112 bytes	3632 bytes	+ 520 bytes per extra odd power
Scratchpad:	+ 32 bytes (fixed, at end of PKA RAM)		
CRT Moduli	**1 odd power**		**> 1 odd power (add to '1 odd power' sizes)**
2x512 bits	696 bytes		+ 72 bytes per extra odd power
2x1024 bits	1336 bytes		+ 136 bytes per extra odd power
2x2048 bits	2616 bytes		+ 264 bytes per extra odd power
Scratchpad:	+ 48 bytes (fixed, at end of PKA RAM)		

Table 18-12 indicates the maximum number of odd powers that can be used for different standard PKA RAM sizes and PKA Engine with CRT support types (non-CRT operations using PKA_CPTR = PKA_DPTR):

Table 18-12. Maximum Number of Odd Powers for Different PKA RAM Sizes

Operation	Modulus and Exponent Sizes	Maximum Number of Odd Powers for PKA RAM Sizes of:			
		1K Byte	2K Byte	4K Byte	8K Byte
Non-CRT	1024 bits	2	9	16	16
	2048 bits	N/A	2	10	16
	4096 bits	N/A	N/A	2	10
CRT	2x512 bits	4	16	16	16
	2x1024 bits	N/A	5	16	16
	2x2048 bits	N/A	N/A	6	16

Notes:

- Using more than 8 odd powers is not advisable as the speed advantage for each extra odd power decreases rapidly (and can even become negative for short Exponent vector lengths due to the extra pre-processing required).
- The maximum amount of odd powers is 16 (limited by firmware). All '16 odd powers' entries in the table above hit this limit – they are not limited by the PKA RAM size.
- ExpMod-ACT4 uses 8 odd powers and can therefore not be used with minimum PKA RAM configurations.
- PPC460EX/EXr/GT contain 8K Bytes of PKA RAM.

18.1.6 True Random Number Generator

The true random number generator provides a truly random, non-deterministic noise source for the purpose of generating keys, Initialization Vectors (IV's), and other random number requirements.

18.1.7 Pseudo Random Number Generator

The ANSI X9.17 Annex C compliant pseudo random number generator provides a pseudo random source for the purpose of generating Initialization Vectors (IV's) for DES/Triple-DES and AES algorithms to the Packet Engine, and other pseudo random number requirements.

It provides up to 16-bytes of data at a time, enabling the generation of random IVs for DES, Triple-DES and AES on a per packet basis without slowing down the system. The DES block within the PRNG uses a 64-bit LFSR as input data.

Unlike true random number generators, which exploit the randomness that occurs in some physical phenomena, pseudo random number generators are devices or algorithms that output statistically independent and unbiased numbers. In general, a PRNG is a deterministic algorithm that for a truly random binary sequence, outputs a binary sequence which appears to be random.

18.1.8 Interrupt Controller

The interrupt controller provides the ability to initiate descriptor fetches by means a security function input interrupt. This avoids the overhead of frequent polling for valid descriptors across the PLB Master interface.

In addition, under programmable configuration control, additional interrupts may be generated due to completion of certain operations such as *packet processing complete*, *user DMA transfer complete*, *public key accelerator operation complete*, and so on.

The sixteen interrupts that can be delivered to the UIC are shown in *Interrupt Unmasked and Masked Status Registers (CRYP0_INT_UNMSK/MSK)* on page 397.

The CRYP0_INT_EN register provides the mask to select which interrupt source is enabled. A pair of status registers, CRYP0_INT_UNMSK and CRYP0_INT_MSK, allows the processor to read the status of any interrupt source before and after the mask is applied.

These interrupts are latched in both the Unmasked and Masked status registers. Clearing the interrupts using the CRYP0_INT_MSK registers resets both Masked and Unmasked latch stages.

18.1.9 Device Controller

The Device Controller contains miscellaneous identification and cryptographic control registers. They uniquely identify the security function version and its incorporated features to allow software to tune its control to the target instantiation and its capabilities.

18.2 KASUMI Algorithm

The KASUMI algorithm is managed by the Security function.

The KASUMI function includes the following features:
- Key scheduling hardware
- f8 confidentiality and f9 integrity algorithm support
- Automatic data padding mechanism for f9 algorithm
- KASUMI encryption and decryption modes
- 32-bit slave interface
- Fully synchronous to the PLB clock

18.2.1 References

1. *Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 1: f8 and f9 Specification*, V5.0.0, June 2002
2. *Specification of the 3GPP Confidentiality and Integrity Algorithms, Document 2: KASUMI Specification*, V5.0.0, June 2002

18.2.2 Functional Description

The KASUMI function can operate in three modes:
- KASUMI mode (encryption and decryption);
- *f8* mode;
- *f9* mode.

In general, a plain message data is parsed and sequentially fed into KASUMI in 64-bit blocks. The processing of one 64-bit block takes eight rounds. A round takes one clock cycle.

18.2.2.1 KASUMI Mode

In KASUMI mode, the 64-bit plain text data blocks are encrypted or decrypted under a 128-bit key, which is programmed into the KASUMI function. During each round, the Round Keys are derived from this initial key. After eight processing rounds, the KASUMI encrypted data block is stored in the output data register.

When performing encryption, plain text data is transformed into KASUMI cipher text data. When performing decryption, the KASUMI cipher text data is transformed back into the original plain text data.

In order to process data in KASUMI mode, the following KASUMI function registers must be programmed:
- CRYP0_SA_KEY0 :3 (configure KASUMI 128-bit key)
- CRYP0_SA_CMD_0[CALGO] (configure KASUMI encryption/decryption)
- CRYP0_SA_CMD_1[CMODE:31:9:8] (configure KASUMI mode)
- CRYP0_SA_CMD_0[IVLD] (configures IV load location)
- CRYP0_SA_IV0:1 (optional, location of IV value)
- CRYP0_SA_SOURCE (configure pointer to input data)
- CRYP0_SA_DEST (configure pointer to output data)

The data in the SOURCE/DEST buffers have optionally up to 255 bytes of *bypass* data. This data is not processed through the KASUMI algorithm. It bypasses encryption and is copied from the input buffer to the output buffer.

The location of the IV is configurable from either the input buffer or the CRYP0_SA_IV0:1 registers. The IV data could be imbedded in the data in the input buffer, following the optional bypass data but ahead of the payload data.

More information about KASUMI mode can be found in Reference 2.

18.2.2.2 f8 Mode

In *f8* mode, the 64-bit message data blocks are transformed into 64-bit data blocks under the control of a 128-bit Confidentiality Key. The total message length can be up to 2^{16} = 65536 bits = 8191 bytes.

In order to process data in *f8* mode, the following KASUMI registers must be programmed:
- CRYP0_SA_KEY0:3 (configure KASUMI 128-bit Confidentiality key)
- CRYP0_SA_CMD_0[CALGO] (configure KASUMI mode)
- CRYP0_SA_CMD_1[CMODE:31:9:8] (configure KASUMI f8 mode)
- CRYP0_SA_CMD_0[IVLD] (configure IV load location)
- CRYP0_SA_IV0:1 (optional, location of IV value)
- CRYP0_SA_SOURCE (configure pointer to input data)
- CRYP0_SA_DEST (configure pointer to output data)

The data in the SOURCE/DEST buffers have optionally up to 255 bytes of bypass data. This data is not processed through the KASUMI algorithm. It bypasses encryption and is copied from the input buffer to the output buffer.

The location of the IV is configurable from either the input buffer or the CRYP0_SA_IV0:1 registers. The IV data could be imbedded in the data in the input buffer, following the optional bypass data but ahead of the payload data.

More information about the *f8* algorithm can be found in Reference 1.

18.2.2.3 f9 Mode

In *f9* mode, the message data is processed in 64-bit blocks under control of a 128-bit Integrity Key. The result, after all message data has been processed, is a 32-bit Message Authentication Code (MAC). The total length of the message data can be up to 2^{16} = 65536 bits = 8191 bytes.

In order to process data in *f9* mode, the following KASUMI function registers must be programmed:
- CRYP0_SA_IH_D0:3 (configure KASUMI 128-bit Integrity key)
- CRYP0_SA_OH_D0:2 (configure D0[31:0] = COUNT, D1[31:0] = FRESH, D2[31:0] = DIRECTION
- CRYP0_SA_CMD_0[CALGO,HALGO] (configure KASUMI f9 mode)
- CRYP0_SA_CMD_0[LHSTT, SHSTT] (configure HASH loading/saving)
- CRYP0_SA_CMD_1[CMODE :31,9,8] (configure KASUMI f9 mode)
- CRYP0_SA_IV0 :1 (IV1[7 :3] = BEARER, IV1[2] = DIRECTION)
- CRYP0_SA_SOURCE (configure pointer to input data)
- CRYP0_SA_DEST (configure pointer to output data)

The data in the SOURCE/DEST buffers have optionally up to 255 bytes of bypass data. This data is not processed through the KASUMI algorithm. It bypasses encryption and is copied from the input buffer to the output buffer.

The location of the IV is configurable from either the input buffer or the CRYP0_SA_IV0:1 registers. The IV data could be imbedded in the data in the input buffer, following the optional bypass data but ahead of the payload data.

KASUMI f9 hash results (64-bit value) is returned in the CRYP0_SA_IH_D0:1 registers. D0[31:0] MAC-I 32-bit result, D1[31:0] returns the remaining 32-bits.

CRYP0_SA_HASH_B0 returns the final byte count processed by the KASUMI f9 algorithm.

More information about the *f9* algorithm can be found in Reference 1.

18.3 Programing Examples

The following pseudo code examples show the actions that are typically executed by the software to start a cipher session in KASUMI.

18.3.1 KASUMI Mode Example

```
// wait until input buffer available for writing by host
wait KASU0_STAT[6:1] == '000000' (or inbuf_rfd == '1')

write KASU0_KEY0
write KASU0_KEY1
write KASU0_KEY2
write KASU0_KEY3

write KASU0_DATAIN0
write KASU0_DATAIN1

write KASU0_MODE[MDKAS] == '1' // select KASUMI mode
write KASU0_MODE[CRYPT]// select encryption or decryption
```

```
write KASU0_CTRL[6:0]          // indicate available data, e.g. 0x0E if Mode,
                               // Key and Data were written

// wait until output data ready
wait KASU0_STAT[OBUFF] == '1' (or outbuf_data_av == '1')

read KASU0_DATAOUT0
read KASU0_DATAOUT1

write KASU0_CTRL[OBUFF] = '1' // indicate finished reading output data
```

18.3.2 *f8* Mode Example

```
// wait until input buffer available for writing by host
wait KASU0_STAT[6:1] == '000000' (or inbuf_rfd == '1')

write KASU0_KEY0
write KASU0_KEY1
write KASU0_KEY2
write KASU0_KEY3

write KASU0_COUNT
write KASU0_CONFIG

write KASU0_DATAIN0
write KASU0_DATAIN1

write KASU0_MODE[MDF8] == '1' // select f8 mode
write KASU0_MODE[CRYPT] == '1' // select encrypt mode

write KASU0_CTRL[6:0]          // indicate available data, e.g., 0x3E if Mode,
                               // Key, Data, Config and Count were written

// wait until input buffer available for writing by host
wait KASU0_STAT[6:1] == '000000' (or inbuf_rfd == '1')

// Place next data block in pipeline
write KASU0_DATAIN0
write KASU0_DATAIN1

write KASU0_CTRL[6:0]         // indicate available data, e.g., 0x08 if Data
                              // was written
```

```
// wait until output data ready
wait KASU0_STAT[OBUFF] == '1' (or outbuf_data_av == '1')

// Read 1st cipher result
read KASU0_DATAOUT0
read KASU0_DATAOUT1

write KASU0_CTRL[OBUFF] = '1'// indicate finished reading output data

// wait until output data ready
wait KASU0_STAT[OBUFF] == '1' (or outbuf_data_av == '1')

// Read 2nd cipher result
read KASU0_DATAOUT0
read KASU0_DATAOUT1

write KASU0_CTRL[OBUFF] = '1'  // indicate finished reading output data
```

18.3.3 f9 Mode Example

```
// wait until input buffer available for writing by host
wait KASU0_STAT[6:1] == '000000' (or inbuf_rfd == '1')

write KASU0_KEY0
write KASU0_KEY1
write KASU0_KEY2
write KASU0_KEY3

write KASU0_COUNT
write KASU0_FRESH
write KASU0_CONFIG

write KASU0_DATAIN0
write KASU0_DATAIN1

write KASU0_MODE[MDF9] == '1'       // select f9 mode
write KASU0_MODE[CRYPT] == '1'     // select encrypt mode

write KASU0_CTRL[6:0]       // indicate available data, e.g., 0x7E if Mode,
                            // Key, Data, Config, Count and Fresh were written

// wait until input buffer available for writing by host
wait KASU0_STAT[6:1] == '000000' (or inbuf_rfd == '1')
```

```
// Place next data block in pipeline
write KASU0_DATAIN0
write KASU0_DATAIN1

write KASU0_CTRL[6:0]    // indicate available data, e.g., 0x08 if Data
                         // was written

// wait until output data ready
wait KASU0_STAT[OBUFF] == '1' (or outbuf_data_av == '1')

// Read 32-bit MAC result
read KASU0_DATAOUT0

write KASU0_CTRL[OBUFF] = '1'  // indicate finished reading output data
```

18.4 Register Interface

The following sections describe the security function register interface in detail. All of the security function registers are 32-bit MMIO registers.

18.4.1 Byte Ordering

Security function registers are implemented and described in little endian format. Therefore, bit 31 is the most significant bit (msb) and bit 0 is the least significant bit (lsb). Big endian software must consider this when accessing the registers. For example, to put the value 44332211 in a security function 32-bit register, big endian software must either do a store (**stw**) of the value 11223344 or do a byte-reversed store (**stwbr**) of the value 44332211. In either case, the value 44332211 appears on the appropriate word of the PLB data bus. Refer to the *Storage Addressing* section of the *PPC440 Processor User's Manual* for a more detailed description of this process.

18.4.2 Register Summary

Table 18-13 contains a summary list of all the registers associated with the PPC460EX/EXr/GT security function. In some cases, multiple addresses are specified for a register with the distinction of *rev1* or *rev2*. The address to be used is based on the setting of SAREV (bit 14) in CRYP0_SA_CMD_1.

.

Table 18-13. Security Function Registers

Mnemonic	Register	Address	Access	Page
Security Function Control Register				
SDR0_CRYPTO_CFG1	EIP-94 Configuration 1 Register	0x4500	R/W	343
SDR0_CRYPTO_CFG2	EIP-94 Configuration 2 Register	0x4501	R/W	343
SDR0_PKP_CFG1	EIP-PKP Configuration 1 Register	0x4502	R/W	343

Table 18-13. Security Function Registers (Continued)

Mnemonic	Register	Address	Access	Page
SDR0_PKP_CFG2	EIP-PKP Configuration 2 Register	0x4503	R/W	344
Packet Engine (PE) Command Registers				
CRYP0_BYTE_ORDER_CFG	Byte Order Configuration	0x4 001E 00D8	R/W	344
CRYP0_PE_CTLST	PE Control/Status	0x4 0018 0000	R/W	345
CRYP0_PE_SOURCE	PE Source Address	0x4 0018 0004	R/W	351
CRYP0_PE_DEST	PE Destination Address	0x4 0018 0008	R/W	351
CRYP0_PE_SA	PE SA Address	0x4 0018 000C	R/W	352
CRYP0_PE_SA_LEN	PE SA Length	0x4 0018 0010	R/W	352
CRYP0_PE_LENGTH	PE Length	0x4 0018 0010 0x4 0018 0014	R/W	353
Packet Engine Control Registers				
CRYP0_PE_DMA_CF	PE DMA Configuration	0x4 0018 0040	See register	354
CRYP0_PE_DMA_ST	PE DMA Status	0x4 0018 0044	Read only	356
CRYP0_PE_PDR_BA	PE Packet Descriptor Ring Base Address	0x4 0018 0048	R/W	357
CRYP0_PE_RDR_BA	PE Result Descriptor Ring Base Address	0x4 0018 004C	R/W	357
CRYP0_PE_RING_S	PE Ring Size and Offset	0x4 0018 0050	R/W	358
CRYP0_PE_RING_P	PE Ring Poll	0x4 0018 0054	R/W	359
CRYP0_PE_I_RING	PE Internal Ring Status	0x4 0018 0058	Read only	360
CRYP0_PE_E_RING	PE External Ring Status	0x4 0018 005C	Read only	360
CRYP0_PE_IO_THR	PE I/O Threshold	0x4 0018 0060	R/W	361
CRYP0_PE_GATH	PE Gather Particle Ring Base Address	0x4 0018 0064	R/W	361
CRYP0_PE_SCAT	PE Scatter Particle Ring Base Address	0x4 0018 0068	R/W	361
CRYP0_PE_PT_S	PE Particle Ring Size	0x4 0018 006C	R/W	362
CRYP0_PE_PT_CFG	PE Particle Ring Configuration	0x4 0018 0070	R/W	362
CRYP0_PE_PDR_UADDR	PE Packet Descriptor Ring Upper Address	0x4 0018 0080	R/W	362
CRYP0_PE_RDR_UADDR	PE Result Descriptor Ring Upper Address	0x4 0018 0084	R/W	363
CRYP0_PE_PKT_SRC_UADDR	PE Packet Source Upper Address	0x4 0018 0088	R/W	363
CRYP0_PE_PKT_DEST_UADDR	PE Packet Destination Upper Address	0x4 0018 008C	R/W	363
CRYP0_PE_SA_UADDR	PE Security Association Upper Address	0x4 0018 0090	R/W	364
CRYP0_PE_GATH_RING_UADDR	PE Gather Ring Upper Address	0x4 0018 00A0	R/W	364
CRYP0_PE_SCAT_RING_UADDR	PE Scatter Ring Upper Address	0x4 0018 00A4	R/W	364

Table 18-13. Security Function Registers (Continued)

Mnemonic	Register	Address	Access	Page
Packet Engine Security Association (SA) and DMA Registers				
CRYP0_PE_PR_SCA	PE Particle Descriptor Source Address	0x4 0018 0500	R/W (**Note**)	365
CRYP0_PE_PR_SCC	PE Particle Descriptor Source Control	0x4 0018 0504	R/W (**Note**)	365
CRYP0_PE_PR_DTA	PE Particle Descriptor Destination Address	0x4 0018 0580	R/W (**Note**)	366
CRYP0_PE_PR_DTC	PE Particle Descriptor Destination Control	0x4 0018 0584	R/W (**Note**)	366
Note: Although this is a R/W register, in normal operation, the processor does not need to read or write it. Writing is not recommended since it can interfere with correct gather processing. It should be used for debug only.				
Cryptographic Context (SA) Registers				
CRYP0_SA_CMD_0	SA Command 0 Word	0x4 0019 0600	Write only	367
CRYP0_SA_CMD_1	SA Command 1 Word	0x4 0019 0604	Write only	371
CRYP0_SA_KEY0	SA Key 0	0x4 0019 0610	R/W	375
CRYP0_SA_KEY1	SA Key 1	0x4 0019 0614	R/W	375
CRYP0_SA_KEY2	SA Key 2	0x4 0019 0618	R/W	375
CRYP0_SA_KEY3	SA Key 3	0x4 0019 061C	R/W	375
CRYP0_SA_KEY4	SA Key 4	0x4 0019 0620	R/W	375
CRYP0_SA_KEY5	SA Key 5	0x4 0019 0624	R/W	375
CRYP0_SA_KEY6	SA Key 6	0x4 0019 0628	R/W	375
CRYP0_SA_KEY7	SA Key 7	0x4 0019 062C	R/W	375
CRYP0_SA_IH_D0	SA Inner Hash Digest 0	rev1: 0x4 0019 0630	R/W	375
CRYP0_SA_IH_D1	SA Inner Hash Digest 1	rev1: 0x4 0019 0634	R/W	375
CRYP0_SA_IH_D2	SA Inner Hash Digest 2	rev1: 0x4 0019 0638	R/W	375
CRYP0_SA_IH_D3	SA Inner Hash Digest 3	rev1: 0x4 0019 063C	R/W	375
CRYP0_SA_IH_D4	SA Inner Hash Digest 4	rev1: 0x4 0019 0640	R/W	375
CRYP0_SA_IH_D5	SA Inner Hash Digest 5	rev2: 0x4 0019 0644	R/W	375
CRYP0_SA_IH_D6	SA Inner Hash Digest 6	rev2: 0x4 0019 0648	R/W	375
CRYP0_SA_IH_D7	SA Inner Hash Digest 7	rev2: 0x4 0019 064C	R/W	375
CRYP0_SA_IH_D8	SA Inner Hash Digest 8	rev2: 0x4 0019 0650	R/W	375
CRYP0_SA_IH_D9	SA Inner Hash Digest 9	rev2: 0x4 0019 0654	R/W	375
CRYP0_SA_IH_D10	SA Inner Hash Digest 10	rev2: 0x4 0019 0658	R/W	375
CRYP0_SA_IH_D11	SA Inner Hash Digest 11	rev2: 0x4 0019 065C	R/W	375
CRYP0_SA_IH_D12	SA Inner Hash Digest 12	rev2: 0x4 0019 0660	R/W	375
CRYP0_SA_IH_D13	SA Inner Hash Digest 13	rev2: 0x4 0019 0664	R/W	375
CRYP0_SA_IH_D14	SA Inner Hash Digest 14	rev2: 0x4 0019 0668	R/W	375
CRYP0_SA_IH_D15	SA Inner Hash Digest 15	rev2: 0x4 0019 066C	R/W	375

Table 18-13. Security Function Registers (Continued)

Mnemonic	Register	Address	Access	Page
CRYP0_SA_OH_D0	SA Outer Hash Digest 0	rev1: 0x4 0019 0644 rev2: 0x4 0019 0670	R/W	376
CRYP0_SA_OH_D1	SA Outer Hash Digest 1	rev1: 0x4 0019 0648 rev2: 0x4 0019 0674	R/W	376
CRYP0_SA_OH_D2	SA Outer Hash Digest 2	rev1: 0x4 0019 064C rev2: 0x4 0019 0678	R/W	376
CRYP0_SA_OH_D3	SA Outer Hash Digest 3	rev1: 0x4 0019 0650 rev2: 0x4 0019 067C	R/W	376
CRYP0_SA_OH_D4	SA Outer Hash Digest 4	rev1: 0x4 0019 0654 rev2: 0x4 0019 0680	R/W	376
CRYP0_SA_OH_D5	SA Outer Hash Digest 5	rev2: 0x4 0019 0684	R/W	376
CRYP0_SA_OH_D6	SA Outer Hash Digest 6	rev2: 0x4 0019 0688	R/W	376
CRYP0_SA_OH_D7	SA Outer Hash Digest 7	rev2: 0x4 0019 068C	R/W	376
CRYP0_SA_OH_D8	SA Outer Hash Digest 8	rev2: 0x4 0019 0690	R/W	376
CRYP0_SA_OH_D9	SA Outer Hash Digest 9	rev2: 0x4 0019 0694	R/W	376
CRYP0_SA_OH_D10	SA Outer Hash Digest 10	rev2: 0x4 0019 0698	R/W	376
CRYP0_SA_OH_D11	SA Outer Hash Digest 11	rev2: 0x4 0019 069C	R/W	376
CRYP0_SA_OH_D12	SA Outer Hash Digest 12	rev2: 0x4 0019 06A0	R/W	376
CRYP0_SA_OH_D13	SA Outer Hash Digest 13	rev2: 0x4 0019 06A4	R/W	376
CRYP0_SA_OH_D14	SA Outer Hash Digest 14	rev2: 0x4 0019 06A8	R/W	376
CRYP0_SA_OH_D15	SA Outer Hash Digest 15	rev2: 0x4 0019 06AC	R/W	376
CRYP0_SA_SPI	SA IPsec SPI	rev1: 0x4 0019 0658 rev2: 0x4 0019 06B0	R/W	377
CRYP0_SA_SEQ0	SA IPsec Sequence Number 0	rev1: 0x4 0019 065C rev2: 0x4 0019 06B4	R/W	377
CRYP0_SA_SEQ1	SA IPsec Sequence Number 1	rev2: 0x4 0019 06B8	R/W	377
CRYP0_SA_SEQMK0	SA IPsec Sequence Number Mask 0	rev1: 0x4 0019 0660 rev2: 0x4 0019 06BC	R/W	378
CRYP0_SA_SEQMK1	SA IPsec Sequence Number Mask 1	rev1: 0x4 0019 0664 rev2: 0x4 0019 06C0	R/W	378
CRYP0_SA_SEQMK2	SA IPsec Sequence Number Mask 2	rev2: 0x4 0019 06C4	R/W	378
CRYP0_SA_SEQMK3	SA IPsec Sequence Number Mask 3	rev2: 0x4 0019 06C8	R/W	378
CRYP0_SA_PNTR	SA Pointer	rev1: 0x4 0019 066C rev2: 0x4 0019 06DC	R/W	378
CRYP0_SA_ARC4IJ	SA ARC4 i and j Pointer	rev1: 0x4 0019 0670 rev2: 0x4 0019 06E0	R/W	379
CRYP0_SA_ARC4SB	SA ARC4 State Address Pointer	rev1: 0x4 0019 0674 rev2: 0x4 0019 06E4	R/W	379
CRYP0_SA_NOUNCE or CRYP0_SA_IV_0	SA Nounce Value or SA Initialization Vector 0	rev1: 0x4 0019 0668 rev2: 0x4 0019 06CC	R/W	380
CRYP0_SA_IV_1	SA Initialization Vector 1	rev2: 0x4 0019 06D0	R/W	380

Table 18-13. Security Function Registers (Continued)

Mnemonic	Register	Address	Access	Page
CRYP0_SA_IV_2	SA Initialization Vector 2	rev2: 0x4 0019 06D4	R/W	380
CRYP0_SA_IV_3	SA Initialization Vector 3	rev2: 0x4 0019 06D8	R/W	380
CRYP0_SA_HASH_B0	SA Hash Byte Count 0	rev1: 0x4 0019 05D0 rev2: 0x4 0019 0710	R/W	380
CRYP0_SA_HASH_B1	SA Hash Byte Count 1	rev2: 0x4 0019 0714	R/W	380
CRYP0_SA_IH_0	SA Inner Hash 0 (mirror of CRYP0_SA_IH_D0)	rev1: 0x4 0019 06D4 rev2: 0x4 0019 0718	R (master and slave) W (master only	381
CRYP0_SA_IH_1	SA Inner Hash 1 (mirror of CRYP0_SA_IH_D1)	rev1: 0x4 0019 06D8 rev2: 0x4 0019 071C	R (master and slave) W (master only	381
CRYP0_SA_IH_2	SA Inner Hash 2 (mirror of CRYP0_SA_IH_D2)	rev1: 0x4 0019 06DC rev2: 0x4 0019 0720	R (master and slave) W (master only	381
CRYP0_SA_IH_3	SA Inner Hash 3 (mirror of CRYP0_SA_IH_D3)	rev1: 0x4 0019 06E0 rev2: 0x4 0019 0724	R (master and slave) W (master only	381
CRYP0_SA_IH_4	SA Inner Hash 4 (mirror of CRYP0_SA_IH_D4)	rev1: 0x4 0019 06E4 rev2: 0x4 0019 0728	R (master and slave) W (master only	381
CRYP0_SA_IH_5	SA Inner Hash 5 (mirror of CRYP0_SA_IH_D5)	0x4 0019 072C	R (master and slave) W (master only	381
CRYP0_SA_IH_6	SA Inner Hash 6 (mirror of CRYP0_SA_IH_D6)	0x4 0019 0730	R (master and slave) W (master only	381
CRYP0_SA_IH_6	SA Inner Hash 7 (mirror of CRYP0_SA_IH_D7)	0x4 0019 0734	R (master and slave) W (master only	381
CRYP0_SA_IH_8	SA Inner Hash 8 (mirror of CRYP0_SA_IH_D8)	0x4 0019 0738	R (master and slave) W (master only	381
CRYP0_SA_IH_9	SA Inner Hash 9 (mirror of CRYP0_SA_IH_D9)	0x4 0019 073C	R (master and slave) W (master only	381
CRYP0_SA_IH_10	SA Inner Hash 10 (mirror of CRYP0_SA_IH_D10)	0x4 0019 0740	R (master and slave) W (master only	381
CRYP0_SA_IH_11	SA Inner Hash 11 (mirror of CRYP0_SA_IH_D11)	0x4 0019 0744	R (master and slave) W (master only	381

Table 18-13. Security Function Registers (Continued)

Mnemonic	Register	Address	Access	Page
CRYP0_SA_IH_12	SA Inner Hash 12 (mirror of CRYP0_SA_IH_D12)	0x4 0019 0748	R (master and slave) W (master only	381
CRYP0_SA_IH_13	SA Inner Hash 13 (mirror of CRYP0_SA_IH_D13)	0x4 0019 074C	R (master and slave) W (master only	381
CRYP0_SA_IH_14	SA Inner Hash 14 (mirror of CRYP0_SA_IH_D14)	0x4 0019 0750	R (master and slave) W (master only	381
CRYP0_SA_IH_15	SA Inner Hash 15 (mirror of CRYP0_SA_IH_D15)	0x4 0019 0754	R (master and slave) W (master only	381
CRYP0_SA_ICV_0	SA ICV 0—HMAC result (outbound and inbound)	rev1: 0x4 0019 06E8 rev2: 0x4 0019 0758	R/W	381
CRYP0_SA_ICV_1	SA ICV 1—HMAC result (outbound and inbound)	rev1: 0x4 0019 06EC rev2: 0x4 0019 075C	R/W	381
CRYP0_SA_ICV_2	SA ICV 2—HMAC result (outbound and inbound)	rev1: 0x4 0019 06F0 rev2: 0x4 0019 0760	R/W	381
CRYP0_SA_ICV_3	SA ICV 3—HMAC result (outbound and inbound)	rev1: 0x4 0019 06F4 rev2: 0x4 0019 0764	R/W	381
CRYP0_SA_ICV_4	SA ICV 4—HMAC result (outbound and inbound)	rev1: 0x4 0019 06F8 rev2: 0x4 0019 0768	R/W	381
CRYP0_SA_ICV_5	SA ICV 5—HMAC result (outbound and inbound)	rev2: 0x4 0019 076C	R/W	381
CRYP0_SA_ICV_6	SA ICV 6—HMAC result (outbound and inbound)	rev2: 0x4 0019 0770	R/W	381
CRYP0_SA_ICV_7	SA ICV 7—HMAC result (outbound and inbound)	rev2: 0x4 0019 0774	R/W	381
CRYP0_SA_ICV_8	SA ICV 8—HMAC result (outbound and inbound)	rev2: 0x4 0019 0778	R/W	381
CRYP0_SA_ICV_9	SA ICV 9—HMAC result (outbound and inbound)	rev2: 0x4 0019 077C	R/W	381
CRYP0_SA_ICV_10	SA ICV 10—HMAC result (outbound and inbound)	rev2: 0x4 0019 0780	R/W	381
CRYP0_SA_ICV_11	SA ICV 11—HMAC result (outbound and inbound)	rev2: 0x4 0019 0784	R/W	381
CRYP0_SA_ICV_12	SA ICV 12—HMAC result (outbound and inbound)	rev2: 0x4 0019 0788	R/W	381
CRYP0_SA_ICV_13	SA ICV 13—HMAC result (outbound and inbound)	rev2: 0x4 0019 078C	R/W	381
CRYP0_SA_ICV_14	SA ICV 14—HMAC result (outbound and inbound)	rev2: 0x4 0019 0790	R/W	381
CRYP0_SA_ICV_15	SA ICV 15—HMAC result (outbound and inbound)	rev2: 0x4 0019 0794	R/W	381

Table 18-13. Security Function Registers (Continued)

Mnemonic	Register	Address	Access	Page
True Random Number Generator (TRNG) Registers				
CRYP0_TRNG_DATA	TRNG Output	0x4 0011 0000	R/W	382
CRYP0_TRNG_STAT	TRNG Status	0x4 0011 0004	Read only	382
CRYP0_TRNG_CTRL	TRNG Test Control	0x4 0011 0008	R/W	383
CRYP0_TRNG_CFG	TRNG Configuration	0x4 0011 000C	R/W	386
CRYP0_TRNG_ALRM	TRNG Alarm Count	0x4 0011 0010	Read only	385
CRYP0_TRNG_CNTR	TRNG Counter	0x4 0011 0014	R/W	385
CRYP0_TRNG_ENTA	TRNG Test Entropy A	0x4 0011 0018	R/W	384
CRYP0_TRNG_ENTB	TRNG Test Entropy B	0x4 0011 001C	R/W	384
CRYP0_TRNG_X0	TRNG Test Seed 0	0x4 0011 0020	R/W	385
CRYP0_TRNG_X1	TRNG Test Seed 1	0x4 0011 0024	R/W	385
CRYP0_TRNG_X2	TRNG Test Seed 2	0x4 0011 0028	R/W	385
CRYP0_TRNG_LF0L	TRNG Test Read of LFSR 0 Low	0x4 0011 002C	Read only	386
CRYP0_TRNG_LF0H	TRNG Test Read of LFSR0 High	0x4 0011 0030	Read only	386
CRYP0_TRNG_LF1L	TRNG Test Read of LFSR1 Low	0x4 0011 0034	Read only	387
CRYP0_TRNG_LF1H	TRNG Test Read of LFSR1 High	0x4 0011 0038	Read only	387
CRYP0_TRNG_K0_L	TRNG Triple DES Key 0 Low	0x4 0011 003C	Write only	387
CRYP0_TRNG_K0_H	TRNG Triple DES Key 0 High	0x4 0011 0040	Write only	387
CRYP0_TRNG_K1_L	TRNG Triple DES Key 1 Low	0x4 0011 0044	Write only	387
CRYP0_TRNG_K1_H	TRNG Triple DES Key 1 High	0x4 0011 0048	Write only	387
CRYP0_TRNG_IV_L	TRNG Initialization Vector Low	0x4 0011 004C	Write only	388
CRYP0_TRNG_IV_H	TRNG Initialization Vector High	0x4 0011 0050	Write only	388
Public Key Accelerator (PKA) Registers				
CRYP0_PKA_A_PTR	PKA A Vector Address	0x4 0011 4000	R/W	388
CRYP0_PKA_B_PTR	PKA B Vector Address	0x4 0011 4004	R/W	388
CRYP0_PKA_C_PTR	PKA C Vector Address	0x4 0011 4008	R/W	388
CRYP0_PKA_D_PTR	PKA D Vector Address	0x4 0011 400C	R/W	388
CRYP0_PKA_A_LEN	PKA A Vector Length	0x4 0011 4010	R/W	388
CRYP0_PKA_B_LEN	PKA B Vector Length	0x4 0011 4014	R/W	388
CRYP0_PKA_SHIFT	PKA Shift	0x4 0011 4018	R/W	389
CRYP0_PKA_FUNC	PKA Function Code	0x4 0011 401C	R/W	389
CRYP0_PKA_COMP	PKA Comparison Result	0x4 0011 4020	Read only	393
CRYP0_PKA_DIV	PKA Address of Quotient MSW	0x4 0011 4024	Read only	394
CRYP0_PKA_MOD	PKA Address of Remainder MSW	0x4 0011 4028	Read only	395

Table 18-13. Security Function Registers (Continued)

Mnemonic	Register	Address	Access	Page
CRYP0_PKA_SEQ	PKA Sequencer Control/Status	0x4 0011 40C8	See register	395
Interrupt Controller Registers				
CRYP0_INT_UNMSK	Interrupt Unmasked Status	0x4 001D 00A0	Read only	397
CRYP0_INT_MSK	Interrupt Masked Status	0x4 001D 00A4	R/W	397
CRYP0_INT_EN	Interrupt Mask	0x4 001D 00A8	R/W	398
CRYP0_INT_CFG	Interrupt Configuration	0x4 001D 00AC	R/W	399
CRYP0_INT_DESRD	Interrupt Force Descriptor Read	0x4 001D 00B0	Write only	400
CRYP0_INT_DESCT	Interrupt Descriptor Count	0x4 001D 00B4	R/W	400
CRYP0_INT_TMO	Interrupt Timeout Count	0x4 001D 00B8	R/W	401
Device Controller Registers				
CRYP0_DC_CTRL	Device Control	0x4 001E 0080	R/W	401
CRYP0_DC_DEVID	Device ID	0x4 001E 0084	Read only	402
CRYP0_DC_DEVINF	Device Information	0x4 001E 0088	Read only	402
DMA Controller Registers				
CRYP0_DMA_USRC	DMA Source Address	0x4 001E 0094	R/W	403
CRYP0_DMA_UDST	DMA Destination Address	0x4 001E 0098	R/W	403
CRYP0_DMA_UCMD	DMA Command	0x4 001E 009C	R/W	403
CRYP0_DMA_CFG	DMA Configuration/Status	0x4 001E 00D4	R/W	404
Pseudo Random Number Generator (PRNG) Registers				
CRYP0_PRNG_STAT	PRNG Status	0x4 001F 0000	Read only	404
CRYP0_PRNG_CTRL	PRNG Control	0x4 001F 0004	R/W	405
CRYP0_PRNG_SDL	PRNG Seed Value Low	0x4 001F 0008	Write only	405
CRYP0_PRNG_SDH	PRNG Seed Value High	0x4 001F 000C	Write only	405
CRYP0_PRNG_K0L	PRNG Key 0 Low	0x4 001F 0010	Write only	406
CRYP0_PRNG_K0H	PRNG Key 0 High	0x4 001F 0014	Write only	406
CRYP0_PRNG_K1L	PRNG Key 1 Low	0x4 001F 0018	Write only	406
CRYP0_PRNG_K1H	PRNG Key 1 High	0x4 001F 001C	Write only	406
CRYP0_PRNG_RS0	PRNG Result 0 (31:0)	0x4 001F 0020	Read only	406
CRYP0_PRNG_RS1	PRNG Result 1 (63:32)	0x4 001F 0024	Read only	406
CRYP0_PRNG_RS2	PRNG Result 2 (95:64)	0x4 001F 0028	Read only	406
CRYP0_PRNG_RS3	PRNG Result 3 (127:96)	0x4 001F 002C	Read only	406
CRYP0_PRNG_LFL	PRNG LFSR Low	0x4 001F 0030	R/W	407
CRYP0_PRNG_LFH	PRNG LFSR High	0x4 001F 0034	R/W	407

18.4.3 EIP-94 Configuration Register 1 (SDR0_CRYPTO_CFG1)

Figure 18-2. EIP-94 Configuration Register 1 (SDR0_CRYPTO_CFG1)

Bit	Mnemonic	Description	Comments
0	INT	DESC Command 0 No action 1 Initiate descriptor fetches in the Packet Engine	This bit must be cleared to 0 after being used.
1:18		Reserved	
19:31	AMASK	CRYPTO base address low Bits 32:44 of 64-bit base address of the Security Function.	Reset value = 0b0000000000011.

18.4.4 EIP-94 Configuration Register 2 (SDR0_CRYPTO_CFG2)

Figure 18-3. EIP-94 Configuration Register 2 (SDR0_CRYPTO_CFG2)

Bit	Mnemonic	Description	Comments
0:31	AMASK	CRYPTO base address high Bits 0:31 of 64-bit base address of the Security Function.	Reset value = 0x0000_0004.

18.4.5 EIP-PKA Configuration Register (SDR0_PKP_CFG1)

Figure 18-4. EIP-PKP Control Register (SDR0_PKP_CFG1)

Bit	Mnemonic	Description	Comments
0:1		Reserved	
2:3	BOC	Byte order control 00 No action 01 Reverse byte order within a half-word 10 Reverse byte order within a word 11 Reserved	
4:15		Reserved	
16:31	AMASK	PKB base address low Bits 32:47 of 64-bit base address of the PKA and TRNG functions.	Reset value = 0x0011.

18.4.6 EIP-PKP Configuration Register 2 (SDR0_PKP_CFG2)

Figure 18-5. EIP-PKP Configuration Register 2 (SDR0_PKP_CFG2)

Bit	Mnemonic	Description	Comments
0:31	AMASK	PKP base address high Bits 0:31 of 64-bit base address of the PKA and TRNG functions.	Reset value = 0x0000_0004.

18.4.7 Byte Order Configuration Register (CRYP0_BYTE_ORDER_CFG)

Reset: 0x0000 0000

Figure 18-6. Byte Order Configuration Register (CRYP0_BYTE_ORDER_CFG)

Bit	Mnemonic	Description	Comments
31:17		Reserved	
17:16	TD	Byte order for Target Data on PLB Slave interface 00 No action 01 Reverse bytes within a half-word 10 Reverse bytes within a word 11 Reserved	
15:12	SGPD	Byte order Scatter/Gather Particle Descriptor Same as bits 0:3.	Same as bits 0:3.
11:8	DATA	Byte order for packet data Same as bits 0:3.	Same as bits 0:3.
7:4	SA	Byte order for Security Association (SA) Same as bits 0:3.	Same as bits 0:3.
3:0	PD	Byte order for Packet Descriptor Ring/Result Descriptor Ring 0001 Reverse byte order within a half word 0010 Reverse byte order within a word 0100 Reverse byte order within a dual word 1000 Reverse byte order within a quad word All others are Reserved	Half word = 16 bits. Word = 32 bits. Dual word = 64 bits. Quad word = 128 bits.

18.4.8 PE Control/Status Register (CRYP0_PE_CTLST)

The PE Control/Status Register provides basic command information to the PE. Together with the data pointed to in the SA structure, this provides the PE its instructions for processing a packet. Once the requested operation has completed, successfully or unsuccessfully, this register provides result status. Using this register, the processor can indicate whether the SA of the previous packet is the same as the SA from this packet. If that is the same, the SA does not have to be loaded. The current SA that is already available can be used to process this packet.

Reset: 0x0000 0042

Figure 18-7. PE Control/Status Register (CRYP0_PE_CTLST)

Bit	Mnemonic	Description	Comments
31:24	PCPS	Pad Control/Pad Status This field is used to specify Padding Control and return Padding Status. **Pad Control:** 0000 0000 No padding[1]. Otherwise, align packet end to modulo: 0000 0001 1-byte boundary 0000 0010 4-byte boundary 0000 0X00 8-byte boundary 0000 1000 16-byte boundary 0001 0000 32-byte boundary 0010 0000 64-byte boundary 0100 0000 128-byte boundary 1000 0000 256-byte boundary **Note 1:** For Zero Pad or Constant Pad mode no bytes are inserted. For SSL and TLS Pad at least the pad length field is inserted. For IPSEC Pad, a pad-length field of one byte followed by a Next Header field of one byte is inserted. For IPsec ESP outbound operations the packet is always padded to a 4-byte boundary. So 8B for DES, 16B for AES and 4B for Null Crypto. **Pad Status:** See comments.	**Pad Control:** Allows the Host to specify a Pad boundary for outbound operations. This feature may be used for "traffic flow security" in order to conceal the number of payload bytes in an encrypted packet. This field is ignored for all inbound operations. Basic outbound operations that use Stream Ciphers (ARC4 or AES Counter modes) when Pad Stream Cipher is disabled (CRYP0_SA_CMD_0[17] = 0), and SSL Outbound operations that use Stream Ciphers. **Pad Status:** This field is updated by the security function. For an inbound operation that uses IPsec, PKCS#7, SSL, or TLS Pad modes it returns the number of detected pad bytes. For all other inbound operations it returns zero since the other pad modes do not allow implicit determination of pad count. For an outbound operation it returns the number of inserted pad bytes for all Pad modes. In case of a Pad Verify Failure it returns zero. The Pad Count includes added bytes such as the pad length and Next Header field in an IPsec ESP pad.

Figure 18-7. PE Control/Status Register (CRYP0_PE_CTLST) (Continued)

Bit	Mnemonic	Description	Comments
23:16	S	Status This byte encodes the result status for the operation requested in the corresponding Packet Descriptor. This is a read-only field in the Result Descriptor (it is "don't care" in the Command Descriptor). A write to any of the bits has no effect. The security function has built-in circuitry to detect faults that generate a packet failure condition. If the security function detects a fault, it reports the status in this byte to avoid having corrupted data enter the rest of the system. 0000 0000 No error. Successful completion	The packet is fully processed.
		xxxx xxx1 Authentication Failure For an inbound IPsec operation, the Integrity Check Value (ICV) did not match the computed value. For an inbound SSL or TLS operation, the Message Authentication Code (MAC) did not match the computed value.	The packet is fully processed. The SA is not updated for IPsec operations with header processing enabled, SSL operations and TLS operations.
		xxxx xx1x Pad Verify Failure For an inbound operation using IPsec, PKCS#7 or TLS padding, the decrypted pad did not match the expected values for the selected pad mode.	The packet is fully processed. The SA is not updated for IPsec operations with header processing enabled, TLS operations and SSL operations with TLS padding mode.
		xxxx x1xx Sequence Number Failure On an inbound IPsec operation, it indicates there was a fault in the Anti-Replay Sequence Number. On an outbound IPsec packet, it indicates a sequence number overflow (count has reached 2^{32} and has incremented to 0). The counter will wrap.	The packet is fully processed.
		0000 1xxx Invalid Command. Illegal packet command setting. Valid commands are: IPSEC, SSL, TLS, SRTP, Basic-Encrypt, Basic-Decrypt, Basic Hash, Encrypt-Hash, Hash-Decrypt. All other commands are invalid.	Packet command is ignored and no packet is processed. The packet must be re-queued or discarded.
		0001 1xxx Invalid Algorithm An invalid algorithm setting was selected in the SA-record for Hash or Symmetric Crypto. Valid algorithms are: DES, Triple-DES, ARC4, AES, Null-Crypto, MD5, SHA-1, SHA-2 (224, 256-, 384-, and 512-bit digest) KASUMI, KASUMI f8, KASUMI f9, and Null-Hash. All other algorithms are invalid.	Packet command is ignored and no packet is processed. The packet must be re-queued or discarded.
		0010 1xxx Prohibited Algorithm A prohibited algorithm was selected in the SA-record.Prohibited algorithms are: Triple-DES when Triple-DES is disabled, Basic operations with Null-Hash or Null-Crypto, SSL or TLS with Null-Hash, IPsec ESP with Null-Hash and Null-Crypto and IPsec with ARC4.	Packet command is ignored and no packet is processed. The packet must be re-queued or discarded.

Figure 18-7. PE Control/Status Register (CRYP0_PE_CTLST) (Continued)

Bit	Mnemonic	Description	Comments
23:16 (cont.)	S	0011 1xxx Zero Length. The Packet Length field is zero, which is illegal. For IPsec the ICV is stripped before the length is checked. Any input length < 13 bytes results in an Zero Length Error.	Packet command is ignored and no packet is processed. The packet must be re-queued or discarded.
		0100 1xxx DMA Error A fatal error occurred during a DMA operation. A PLB timeout response occurred on a SA-Record read/write, SA-State read/write, ARC4 State read/write, or Packet Data read/write operation.	This packet is aborted. The packet must be re-queued or discarded. An Interrupt is generated. Scatter ring is reset.
		0101 1xxx Invalid IP Header Incorrect Packet Header for IPsec. Incorrect IP Headers are: - Headers that contain an invalid length in the length-field of the header preceding the AH. - Headers that contain an invalid next header field. - The AH is missing in the packet. - The input data of the AH/ESP packet is less than the IP Header + Extension Headers + AH/ESP Headers + IV, if applicable/available for the protocol. For IPsec the ICV is stripped before the length is checked.	This packet is aborted. The packet must be re-queued or discarded. An Interrupt is generated. The Scatter ring is reset.
		0110 1xxx IPsec SPI Mismatch On an inbound packet the 32-bit SPI value in the packet does not match the value in the SA while header processing is enabled.	The packet is fully processed.
		0111 1xxx Block Size Error The length of the inbound packet is not a multiple of the DES or AES block cipher length. For outbound packets the size is always automatically aligned to the correct block size. The hashed packet length is not a multiple of the hash block size in case of an intermediate hash operation. In case of a final hash operation, no error is generated. For IPsec the ICV is stripped before the Block Size is checked.	Packet is fully processed. An Interrupt is generated. Scatter ring is reset.

Figure 18-7. PE Control/Status Register (CRYP0_PE_CTLST) (Continued)

Bit	Mnemonic	Description	Comments
23:16 (cont.)	S	1000 1xxx Invalid Combination Illegal combination of descriptor and/ or command settings is selected which can cause the PE to hang. Illegal combinations are: - IPSEC ESP with crypto algorithm ARC4. - IPSEC AH with Null-Hash and Copy_Header and Copy_Payload both zero. - SRTP with hash algorithm other than SHA-1. - SRTP with crypto algorithm other than null-crypto or AES-ICM or AES-CTR. - SSL or TLS with load IV from input. - SSL or TLS with crypto algorithm AES-CTR or AES-ICM mode. - SSL with other pad modes than SSL or TLS padding (only for block ciphers!).	Packet command is ignored and no packet is processed. The packet must be re-queued or discarded.
		1001 1xxx Length Error Illegal length detected. Basic operations: Packet data length ≤ Hash/Encrypt Offset SRTP operations: Packet data length ≤ IV (opt.) + Bypass Offset + ROC SSL operations: Packet data length ≤ 11 TLS operations: Packet data length ≤ 13 For IPsec the ICV is stripped before the length is checked. Any input length <12 bytes or non-4-byte aligned lengths result in a Length Error.	Packet command is ignored and no packet is processed. The packet must be re-queued or discarded.
		1010 1xxx Bus Master Read Error An error occurred during a bus Master transfer. This packet is aborted. The packet must be re-queued or discarded. And Interrupt is generated. Scattering is reset.	This packet is aborted. The packet must be re-queued or discarded. An Interrupt is generated. Scatter ring is reset.
		1011 1xxx Bus Master Write Error An error occurred during a bus Master write transfer. This packet is aborted. The packet must be re-queued or discarded. And Interrupt is generated. Scattering is reset.	This packet is aborted. The packet must be re-queued or discarded. An Interrupt is generated. Scatter ring is reset.
		1100 1xxx Proc Error An error occurred during the packet processing. This packet is aborted. The packet must be re-queued or discarded. And Interrupt is generated. Scattering is reset.	This packet is aborted. The packet must be re-queued or discarded. An Interrupt is generated. Scatter ring is reset.

Figure 18-7. PE Control/Status Register (CRYP0_PE_CTLST) (Continued)

Bit	Mnemonic	Description	Comments
23:16 (cont.)	S	1101 1xxx Inbound PE Length Error An incorrect length occurred during inbound processing (e.g., due to internal overflow). Packet is fully processed. 111x 1xxx Reserved	Packet is fully processed.
15:8	NHPV	Next Header/Pad Value This field is used to pass the Next Header value between the processor and the security function. **Command descriptor:** For ESP outbound operations, the processor must populate this field with the value that is to be inserted (as part of the ESP trailer) into the Next Header field of the innermost operation's header. For Basic outbound operations that use Constant Pad mode or Constant SSL Pad mode the processor must specify the fixed constant Pad Value in this field. For Basic outbound operations that use IPsec Pad mode the processor must specify the Next Header value in this field. For SSL outbound operations that use Constant SSL Pad mode the processor must specify the fixed constant Pad Value in this field. For all other outbound operations and all inbound operations this field is not used. **Result descriptor:** For IPsec ESP outbound operations this field returns the decimal value 50. For IPsec AH outbound operations this field returns the decimal value 51. For all other outbound operations the security function will not update this field. For IPsec inbound operations and Basic inbound operations that use IPsec padding mode the security function returns the Next Header field it detects on the innermost operation's header, which will typically be the value for the payload protocol, such as TCP or UDP. However, in bundling scenarios or in IPv6 with destination options, another header value could be seen. For all other inbound operations and in case of an error the returned Pad Value is zero.	
7:6		Reserved	

Figure 18-7. PE Control/Status Register (CRYP0_PE_CTLST) (Continued)

Bit	Mnemonic	Description	Comments
5	CMD	Use Cached SA 0 New SA must be fetched from processor memory 1 SA currently in the PE is used for this packet	When 1, the SA for this packet does not have to be reloaded. All fields used for the previous packet can be reused, including the updated fields. These updates correspond directly to the values that are necessary for this packet. Note that only the SA and the Saved IV of the State Record are cached in the PE. The Saved Inner Digest Registers and the Saved Hash Byte Count of the State Record are not cached. When the Save Hash Digest bit is set the Inner Digest value and Hash Byte Count are overwritten for the next operation with the original Inner Digest and Hash Byte Count value of the SA. The Use Cached SA bit must not be used for chained hash operations where information is passed through the SA State to the next operation. This is the case for all basic hash or protocol operations where the hash state must be saved. This bit always returns the last written value on a read by the processor.
4	CMD	Hash Final 0 Specifies that the PE perform an intermediate hash operation by generating an intermediate hash digest on the data presented. 0 Specifies that the PE append the required final hash pad and generate the final hash digest on the data presented. This completes the hash operation.	This bit is only applicable for Basic Hash, Basic Encrypt-Hash and Basic Hash-Decrypt operations and is not required for IPsec, SSL, TLS or SRTP protocol operations. This bit always returns the last written value on a read by the processor.
3	CMD	Init ARC4 Initialize the ARC4 cryptographic algorithm. 0 The ARC4 initialization depends on the value of CRYP0_SA_CMD1 bit #29, ARC Stateless / Stateful mode. If this bit is set to 1, the ARC4 State Record and i/j pointer are loaded to continue the encrypt/decrypt processing from the previous algorithm state. If this bit is set to 0, the key is read from the SA-record, and the ARC4 S-boxes are initialized using this key, prior to the encryption/decryption of data. 1 Specifies that this is the first packet to be processed with a new key. The key is read from the SA-record, and the ARC4 S-boxes are initialized using this key, prior to the encryption/decryption of data.	This bit is only applicable for ARC4 mode.
2		Reserved	
1	CMD	CryptCore Done 0 The security function has not finished processing this descriptor (or Command Queue) 1 The security function has finished processing this descriptor (or Command Queue) and has returned ownership to the processor.	This bit can be reset by both the Host and the security function, but only the security function can set this bit. This bit always returns 1 on a read by the processor.

Figure 18-7. PE Control/Status Register (CRYP0_PE_CTLST) (Continued)			
Bit	**Mnemonic**	**Description**	**Comments**
0	CMD	Host Ready 0 Processor has not populated the descriptor. 1 Processor has populated the descriptor. The processor function assumes that it can begin processing the descriptor.	This bit can be reset by both the processor and the security function, but only the processor can set this bit. This bit always returns 0 on a read by the processor. Note that this bit is only read by the security function in Autonomous Ring mode; it is ignored when directly writing to the Command Queue (instead, the command is fired by the write of the last word in the Command Queue).

18.4.9 PE Source Address Register (CRYP0_PE_SOURCE)

This register specifies the starting address for the packet to be processed.

Reset: 0x0000 0000

Figure 18-8. PE Source Address Register (CRYP0_PE_SOURCE)			
Bit	**Mnemonic**	**Description**	**Comments**
31:0		Packet Source Address	This address does not have to be on a word boundary. For Gather processing, this register contains a pointer to the first Gather Particle Descriptor (which then points to the first buffer).

18.4.10 PE Destination Address Register (CRYP0_PE_DEST)

This register specifies the starting address at which to write the result data from the requested operation.

Reset: 0x0000 0000

Figure 18-9. PE Destination Address Register (CRYP0_PE_DEST)			
Bit	**Mnemonic**	**Description**	**Comments**
31:0		Packet Destination Address	This address does not have to be on a word boundary. For Scatter processing, the Packet Destination Address is ignored on input. Upon completion of packet processing, this field is updated in the Result Descriptor with a pointer to the first Scatter Particle Descriptor (which then points to the first buffer).

18.4.11 PE SA Address Register (CRYP0_PE_SA)

This register specifies the starting address of the SA-record.

Reset: 0x0000_0000

Figure 18-10. PE SA Address Register (CRYP0_PE_SA)

Bit	Mnemonic	Description	Comments
31:0		SA Address	Generally, this address is on a word boundary. However, it is not mandatory. This register is not used when Use Cached SA is specified.

18.4.12 PE SA Length Register (CRYP0_PE_SA_LEN)

This register specifies the length (in words) of the SA-Record structure, excluding the length of the State Record and ARC4 State.

This register is available only when bit 11, Enable Dynamic SA, in the CRYP0_PE_DMA_CFG register is set to 1. In this case, CRYP0_PE_SA_LEN exists at 0x4 0011 0010 and CRYP0_PE_LENGTH exists at 0x4 0011 0014.

Reset: 0x0000_0000

Figure 18-11. PE SA Length Register (CRYP0_PE_SA_LEN)

Bit	Mnemonic	Description	Comments
31:0	SALEN	SA Length Security Association (SA) length in words.	

18.4.13 PE Length Register (CRYP0_PE_LENGTH)

When performing direct writes to the Command Queue (Descriptor Ring disabled), a write to this register will "fire" the command into the PE. This takes the place of the ownership bits in the command byte in the Descriptor Ring modes. Since this word is the last in the Command Queue, it should be written last, after the other registers have been properly programmed.

This register exists at 0x4 0018 0010 if CRYP0_PE_SA_LEN does not exist. Otherwise it exists at 0x4 0011 0014.

Reset: 0x0000 0000

Figure 18-12. PE Length Register (CRYP0_PE_LENGTH)

Bit	Mnemonic	Description	Comments
31:24	BYP	Bypass Specifies the offset, in words, between the hash data and the encryption/decryption data for SRTP operations.	The Hash Encrypt Offset in CRYP0_SA_CMD_1 has the same function, except that it is part of the SA and, therefore, applies the same value for all packets. Since for SRTP the Hash Encrypt Offset can be variable on a per packet basis, it is part of the descriptor. Valid bypass offsets range from 0 to 255 words.
23	CCD	CryptCore Done Mirrors the equivalent bit in the first Control/ Status word of the descriptor.	Repeated in the last word in order to guarantee ownership consistency between the first and last word of the descriptor. When the descriptor fetch engine reads a descriptor, this ownership bit must match that in the first word, or the descriptor is discarded and fetched again. A write to this bit has no effect.
22	HRDY	Host Ready Mirrors the equivalent bit in the first Control/ Status word of the descriptor.	Repeated in the last word in order to guarantee ownership consistency between the first and last word of the descriptor. When the descriptor fetch engine reads a descriptor, this ownership bit must match that in the first word, or the descriptor is discarded and fetched again. A write to this bit has no effect.
21:20		Reserved	
19:0	L	Length Total length (in bytes) of all data passed to the Input Buffer of the security function for an operation. Valid length range is 1 to 1,048,575 bytes.	A length of 0 bytes is illegal and will result in an error status code in the Result Descriptor. In case of a DMA Error, Block Size Error or Invalid IP Header Error, the Length field returns zero in the Result Descriptor. These bits always return the last written value on a read by the processor except when any of these errors occur: DMA Error Block Size Error Invalid IP Header Error Then it returns zero.

18.4.14 PE DMA Configuration Register (CRYP0_PE_DMA_CF)

This register is used to select static settings that control the packet-processing path. These settings are typically set at initialization and not changed again.

Access: See register fields

Reset: 0x0014 5010

Figure 18-13. PE DMA Configuration Register (CRYP0_PE_DMA_CF)

Bit	Mnemonic	Description	Comments
31:25		Reserved	
24	DHDMAM	Direct Host DMA Mode Determines security function mode. 0 Master Mode 1 Direct Host DMA Mode	In Direct Host DMA Mode (mastering ability is disabled), the processor must write all data to the security function.
23:22		Reserved	
21	BOTDEN	Enable Reverse Byte Order for Target Data 0 No reverse byte order 1 Apply reverse byte order	Enable reverse byte order.
20	SPDROU	Suppress PDR Ownership Update Determines whether the security function updates the ownership bits in the Command Descriptor on the PDR. 0 Clear the ownership bits of the Command Descriptor when it finishes an operation. This prevents the security function from reprocessing an old descriptor when it wraps around the PDR. 1 Do not clear the ownership bits in the packet descriptor when it completes an operation. In this case, the processor is responsible for clearing the ownership bits.	This setting is ignored if the PDR and RDR overlap. The processor must clear these ownership bits before the security function is allowed to wrap entirely around the PDR to reencounter old descriptors. As long as there is at least one non-security function-owned descriptor separating the newest valid descriptor and the oldest, this does not occur. Choosing this setting has the advantage of eliminating a separate DMA write to the PDR. The Result Descriptor is always written by the security function.
19	BOSGPDEN	Reverse Byte Order of S/G Descriptor Enable 0 No reverse byte order 1 Apply reverse byte order	Enables the reverse byte order of Scatter/Gather.
18	BODATAEN	Reverse Byte Order of Data Enable 0 No reverse byte order 1 Apply reverse byte order	Enables the reverse byte order of Data.
17	BOSAEN	Reverse Byte Order of SA Descriptor Enable 0 No reverse byte order 1 Apply reverse byte order	Enables the reverse byte order of SAs.
16	BOPDEN	Reverse Byte Order of Packet Descriptor Enable 0 No reverse byte order 1 Apply reverse byte order	Enables the reverse byte order of packet descriptor.
15:12		Reserved	
11	ENDSA	Enable Dynamic SA 0 Disable Dynamic SA (Fixed length - rev1: 32 words rev2: 58 words) 1 Enable Dynamic SA	This is an SA-record with variable length.

Figure 18-13. PE DMA Configuration Register (CRYP0_PE_DMA_CF) (Continued)

Bit	Mnemonic	Description	Comments
10:9	PDRMODE	Packet Follows Descriptor This bit along with SAPP are used as a 2-bit Descriptor Ring mode selector. They set-up the way that the PE fetches descriptors, SAs, and packet data. 00 Basic Ring Mode - Descriptor, SA, and packet data are located in different memory locations. The descriptor is located in a separate descriptor ring with pointers to input/output packet data and SA-record. 01 SA Packet Mode - SA and packet data are located successively in memory. Only pointers for input and output are provided. 10 Packet Descriptor Ring Mode - Descriptor and packet data are located successively in memory. The complete data blocks (packet descriptor + packet data) have a fixed size. Only an SA-record pointer is required. 11 Packet Descriptor SA Packet Mode - All Descriptor, SA, and packet data are located successively in memory. The complete data blocks (packet descriptor + SA + packet data) have a fized size. No pointers are required	
8	PDRE	PDR Enable Selects how the security function receives commands for the PE. 0 Disable the Packet Descriptor Ring manager in the security function and PE commands must be individually entered 1 Enable the Packet Descriptor Ring (PDR) Engine and fetch PE commands from the Descriptor Ring	In Direct Host DMA mode this bit is forced to 0.
7:3		Reserved	
2	RSCG	Reset Scatter/Gather Cache Controls the reset to the Scatter/Gather state machine 0 Release the Scatter/Gather Cache reset 1 Reset the Scatter/Gather Cache	Resets the pointers to the Particle Descriptors to their base address setting. This reset must be coordinated with the owner of the Particle buffers to ensure that the pointers are synchronized after the reset. For a momentary reset, the processor can set this bit to 1 and back to 0 again with no delay between the sets.
1	RPDRCP	Reset PDR Counters/Pointers Controls reset to the Packet Descriptor Ring state machine. 0 Release the Packet Descriptor Ring state machine reset 1 Reset the Packet Descriptor Ring state machine	Resets the pointers for both the Packet Descriptors and Result Descriptors to their base address setting. If the PDR ring is not enabled, this bit may be left in the reset state. This reset must be coordinated with the owner of the Descriptor Ring to ensure that the pointers are synchronized after the reset. For a momentary reset, the processor can set this bit to 1 and back to 0 again with no delay between the sets.
0	RPE	Reset PE Controls reset to both the PE and the state machine logic that drives header processing, the DMA Controller, and context management. 0 Release the PE reset 1 Reset the PE	For a momentary reset, the processor can set this bit to 1 and back to 0 again with no delay between the sets.

18.4.15 PE DMA Status Register (CRYP0_PE_DMA_ST)

Provides the status of the PE.

Reset: 0x0200 0C03

Figure 18-14. PE DMA Status Register (CRYP0_PE_DMA_ST)

Bit	Mnemonic	Description	Comments
31:22	SEOS	CryptCore Output Size Number of 32-bit words that the PE is requesting be read from the output buffer.	
21:12	SEIS	CryptCore Input Size Number of 32-bit words are available in the PE input buffer.	
11	CQS	Output Request Active Active-low (0) indicates that the PE is requesting that output data be read.	Mirrors the $\overline{\text{OutputRequest}}$ signal that acts as a DMA output request in Direct Host DMA mode.
10		Input Request Active Active-low (0) indicates that the PE is requesting that input data be written.	Mirrors the $\overline{\text{InputRequest}}$ signal that acts as a DMA input request in Direct Host DMA mode.
9		Command Queue Active Indicates that the PE is currently processing a packet.	This bit is 0 when the engine is idle.
8		Reserved	
7	CQS	SPI Mismatch Inbound SPI did not match the expected value supplied in the SA.	
6		ICV Fault Inbound ICV fault detected	The ICV carried within the packet did not match the value just computed.
5		Crypto Pad Fault Inbound cryptographic pad fault detected	Possible faults include Pad Count field not matching the number of Pad bytes detected and Pad field values not matching specified pattern.
4		Outer Hash Done Outer hash processing for this packet is finished	
3		Inner Hash Done Inner hash processing for this packet is finished	
2		Encryption Done Encryption or decryption for this packet is finished	
1		CryptCore Output Done All of the output data for the current packet has been read from the cryptographic output buffer.	
0		CryptCore Input Done Number of bytes specified in the packet descriptor has been written into the cryptographic input buffer.	

18.4.16 PE Packet Descriptor Ring Base Address Register (CRYP0_PE_PDR_BA)

Normally this register contains the Packet Descriptor Ring Base Address (*Figure 18-15*). For diagnostic purposes it can contain the Crypto Length Out (*Figure 18-16*).

Reset: 0x0000 0000

Figure 18-15. PE Packet Descriptor Ring Base Address Register (CRYP0_PE_PDR_BA)

Bit	Mnemonic	Description	Comments
31:0		Packet Descriptor Ring Base Address Allows the processor to specify the base location of the Packet Descriptor Ring in external memory space.	This register is ignored if the CRYP0_PE_RING_S register is set to 0 (indicating that the processor writes directly into descriptor ring).

OR

If in Direct Host DMA mode for read access:

Figure 18-16. PE Packet Descriptor Ring Base Address Register (CRYP0_PE_PDR_BA)

Bit	Mnemonic	Description	Comments
31:20		Reserved	
19:0	CLO	Crypto Length Out Returns the length of the encrypted part of the packet in the PE Output Buffer in bytes.	The encrypted part is the total packet minus the ICV (for IPsec) or TAG (for SRTP). This field should be used for debugging purposes only.

18.4.17 PE Packet Result Ring Base Address Register (CRYP0_PE_RDR_BA)

This register contains the Packet Result Ring Base Address.

If the address of the RDR (Result Descriptor Ring) matches the PDR (Packet Descriptor Ring), then the Packet Engine determines that it is in "Overlaid Descriptor" mode. This means that there is only a single ring used for both commands and results. In this case, the Packet Engine does not have to update the ownership bits in the PDR separately from the RDR, which saves a few DMA cycles.

Reset: 0x0000 0000

Figure 18-17. PE Packet Result Ring Base Address Register (CRYP0_PE_RDR_BA)

Bit	Mnemonic	Description	Comments
31:0		Packet Result Ring Base Address Allows the processor to specify the base location of the Packet Result Ring in external memory space.	This register is applicable even if the CRYP0_PE_RING_S register is set to 0 (indicating that the Host will write directly into command register set). Result Descriptors are always written to a specified ring location. They are never written to the internal Command Queue.

18.4.18 PE Ring Size and Offset Register (CRYP0_PE_RING_S)

This register contains the PE Descriptor Ring Size and Offset.

Reset: 0x0005 0000

Figure 18-18. PE Ring Size and Offset Register (CRYP0_PE_RING_S)

Bit	Mnemonic	Description	Comments
31:16	DRO	Descriptor Ring Offset Offset (in words) between Descriptor Ring entries for an external descriptor ring	This value applies to both the PDR and the RDR. The Descriptor Ring Offset field is only applicable when CRYP0_PE_DMA_CF[PFDES] is 1. When PFDES is 0, the implicit offset must be fixed at five words. This situation occurs after a reset. Settings of 5–65535 represent valid external ring offsets. This offset must allow space for the Packet Descriptor (5W), the packet data, and, optionally, the SA-Record (32W if SAPP is 1). Typical values for the offset would range from 512W (2KB) to 8192W (32KB).
15:10		Reserved	
9:0	DRS	Descriptor Ring Size Descriptor Ring Size (number of entries) for an external descriptor ring	This value applies to both the PDR and the RDR. Settings of 1–1023 represent valid external ring sizes. A setting of 0x000 specifies that an external PDR is not used, and that the processor performs PLB Slave writes of descriptors directly into the Command Queue. This effectively turns off the Descriptor Fetch Engine. In this case, the Result Descriptor Ring is fixed at a size of 1.

18.4.19 PE Ring Poll Register (CRYP0_PE_RING_P)

This register allows programming of two polling parameters that are used in the Descriptor Fetch Engine. The first parameter is the basic polling frequency at which the security function reads a segment of the external PDR into its internal Command Queue.

The second parameter is the retry interval that specifies how long the security function should wait in between retry of reads on an invalid descriptor entry (ownership bit not yet assigned to the security function). The retry Interval should be set to a shorter period than the poll interval. Otherwise, the next poll event will preempt a retry. Both of these parameters are typically used to limit the amount of bus bandwidth that is consumed by the descriptor polling process.

A Read Descriptor interrupt can preempt the poll interval and cause a poll to occur sooner than the timer would dictate. Should this occur, the timer polling interval is reset to its starting point, creating a full delay before the next timed poll.

If descriptors are written directly into the internal Command Queue, this register is ignored. This mode is configured in CRYP0_PE_DMA_CF.

Reset: 0x8000 0000

Figure 18-19. PE Ring Poll Register (CRYP0_PE_RING_P)

Bit	Mnemonic	Description	Comments
31	C	Continuous Specifies that the Descriptor Fetch Engine should continuously read descriptor segments until the Command Queue has a valid descriptor	Once the Command Queue entry becomes free, the security function continuously polls for the next valid descriptor (although the polls will be interleaved with other DMA transfers such as moving packet data in and out of the security function).
30:26			
25:16	RRDIV	Ring Retry Divisor Binary value that is used to divide the main security function clock at the retry frequency.	Values of 1–1023 provide valid retry intervals. A setting of 0x000 disables retries altogether and the security function waits until the next Poll interval occurs. A fixed divisor of 128 is inserted in the main security function clock (PLB clock) prior to the Ring Retry Divisor. **Example:** Main security function clock: 133.33 MHz/128 prescale = 1042 kHz Ring Poll set to 0x001: Frequency = 1042 kHz (0.96us between retries) Ring Poll set to 0x3FF: Frequency = 1018 Hz (982us between retries)
15:12			

Figure 18-19. PE Ring Poll Register (CRYP0_PE_RING_P) (Continued)			
Bit	**Mnemonic**	**Description**	**Comments**
11:0	RPDIV	Ring Poll Divisor Binary value used to divide the main security function clock at the polling frequency	Values of 1–4095 provide valid poll intervals. A setting of 0x000 disables PDR polling altogether. In this case, the Read Descriptor interrupt is the only mechanism for initiating a descriptor poll once an empty descriptor has been encountered. A fixed divisor of 128 is inserted in the main security function clock (PLB clock) prior to the Ring Poll Divisor. Example: Main security function clock: 133.33 MHz/128 prescale = 1042 kHz Ring Poll set to 0x001: Frequency = 1042 kHz (0.96us between polls) Ring Pill set to 0xFFF: Frequency = 254 Hz (3.93ms between polls)

18.4.20 PE Internal Ring Status Register (CRYP0_PE_I_RING)

This register is used to provide real-time status for the internal Packet Descriptor Command Queue. In general, this register is used for debug purposes. But in the case of Command Descriptors being written directly to the Command Queue, it is used to determine how full the Command Queue is.

Reset: 0x0000 0001

Figure 18-20. PE Internal Ring Status Register (CRYP0_PE_I_RING)			
Bit	**Mnemonic**	**Description**	**Comments**
31:1		Reserved	
0	CQ0A	Command Queue 0 Available 0 Queue not available 1 Queue available for a PE command	In the Target Command mode, CQ0A = 1 indicates that the user can write a new packet command into this register.

18.4.21 PE External Ring Status Register (CRYP0_PE_E_RING)

This register is used to provide real-time status for the external PDR (if applicable).

Reset: 0x0000 0000

Figure 18-21. PE External Ring Status Register (CRYP0_PE_E_RING)			
Bit	**Mnemonic**	**Description**	**Comments**
31:26		Reserved	
25:16	ONPD	Offset of Next Packet Descriptor	Indicates the offset from the address in CRYP0_PE_PDR_BA of the Next Packet Descriptor to be read by the PE.
15:0		Reserved	

18.4.22 PE I/O Threshold Register (CRYP0_PE_IO_THR)

This register is used to specify at what "high water" and "low water" points the PE should begin to transfer packet data into or out of the PE buffer RAM. These parameters are useful for controlling the DMA burst size for packet data input and output from the PE.

Reset: 0x0008 0008

Figure 18-22. PE I/O Threshold Register (CRYP0_PE_IO_THR)

Bit	Mnemonic	Description	Comments
31:26		Reserved	
25:16	PEOTH	Output Threshold Number of 32-bit words (1–432) that must be available in the PE Output Data RAM buffer prior to initiating a DMA output transfer	The maximum threshold is 432 instead of 511 bytes since this buffer is also used to store up to 256 pad bytes and 64 bytes of ICV that can be stripped for decryption operations.
15:10		Reserved	
9:0	PEITH	Input Threshold Number of 32-bit words (1–508) that must be available in the PE Input Data RAM buffer prior to initiating a DMA input transfer	The maximum threshold is 508 instead of 511 bytes since this buffer is also used to redirect a 12-byte ICV for IPsec AH operations.

18.4.23 PE Gather Particle Ring Base Address Register (CRYP0_PE_GATH)

This register allows the processor to specify the base location of the Gather Particle Ring.

Reset: 0x0000 0000

Figure 18-23. PE Gather Particle Ring Base Address Register (CRYP0_PE_GATH)

Bit	Mnemonic	Description	Comments
31:0		Gather Particle Ring Source Address	This register is ignored if gather is not enabled in any SA-record.

18.4.24 PE Scatter Particle Ring Base Address Register (CRYP0_PE_SCAT)

This register allows the processor to specify the base location of the Scatter Particle Ring.

Reset: 0x0000 0000

Figure 18-24. PE Scatter Particle Ring Base Address Register (CRYP0_PE_SCAT)

Bit	Mnemonic	Description	Comments
31:0		Scatter Particle Ring Source Address	This register is ignored if scatter is not enabled in any SA-record.

18.4.25 PE Particle Ring Size Register (CRYP0_PE_PT_S)

This register contains the scatter and gather particle ring sizes.

Reset: 0x0000 0000

Figure 18-25. PE Particle Ring Size Register (CRYP0_PE_PT_S)

Bit	Mnemonic	Description	Comments
31:16	SPDRS	Scatter Particle Descriptor Ring Size Number of 2-word entries	Valid sizes range is from 2 to 65532.
15:0	GPDRS	Gather Particle Descriptor Ring Size Number of 2-word entries	Valid sizes range is from 2 to 65532.

18.4.26 PE Particle Ring Configuration Register (CRYP0_PE_PT_CFG)

This register is used to select static settings that control scatter/gather processing. These settings are typically set at initialization and not changed again.

Reset: 0x0000 0000

Figure 18-26. PE Particle Ring Configuration Register (CRYP0_PE_PT_CFG)

Bit	Mnemonic	Description	Comments
31:16		Reserved	
15:0	SPS	Scatter Particle Size Size, in bytes, of each particle in the scatter ring	Valid sizes range is from 4 to 65532 bytes in multiples of four bytes. A non-zero value must be written.

18.4.27 PE Packet Descriptor Ring Upper Address (CRYP0_PE_PDR_UADDR)

This register allows the Host to specify the upper part of the location of the Packet Descriptor Ring in external memory space. The PDR is fetched from: CRYP0_PE_PDR_UADDR||CRYP0_PE_PDR_BA.

Reset: 0x0000 0000

Figure 18-27. PE Packet Descriptor Ring Upper Address (CRYP0_PE_PDR_UADDR)

Bit	Mnemonic	Description	Comments
31:0	PDRUADR	PDR Upper Base Address	

18.4.28 PE Result Descriptor Ring Upper Address (CRYP0_PE_RDR_UADDR)

This register allows the Host to specify the upper part of the location of the Result Descriptor Ring in external memory space. The RDR is fetched from: CRYP0_PE_RDR_UADDR||CRYP0_PE_RDR_BA.

Reset: 0x0000 0000

Figure 18-28. PE Result Descriptor Ring Upper Address (CRYP0_PE_RDR_UADDR)

Bit	Mnemonic	Description	Comments
31:0	RDRUADR	RDR Upper Base Address	

18.4.29 PE Packet Source Upper Address (CRYP0_PE_PKT_SRC_UADDR)

This register allows the Host to specify the upper part of the location of the packet source data in external memory space. The source data is fetched from: CRYP0_PE_PKT_SRC_UADDR||CRYP0_PE_SOURCE.

Reset: 0x0000 0000

Figure 18-29. PE Packet Source Upper Address (CRYP0_PE_PKT_SRC_UADDR)

Bit	Mnemonic	Description	Comments
31:0	SRCUADR	Source Upper Base Address	

18.4.30 PE Packet Destination Upper Address (CRYP0_PE_PKT_DEST_UADDR)

This register allows the Host to specify the upper part of the location of the packet source data in external memory space. The source data is fetched from: CRYP0_PE_PKT_DEST_UADDR||CRYP0_PE_DEST.

Reset: 0x0000 0000

Figure 18-30. PE Packet Destination Upper Address (CRYP0_PE_PKT_DEST_UADDR)

Bit	Mnemonic	Description	Comments
31:0	PKTUADR	Packet Upper Base Address	

18.4.31 PE Security Association Upper Address (CRYP0_PE_SA_UADDR)

This register allows the Host to specify the upper part of the location of the SA in external memory space. The SA is fetched from: CRYP0_PE_SA_UADDR||CRYP0_PE_SA.

Reset: 0x0000 0000

Figure 18-31. PE Security Association Upper Address (CRYP0_PE_SA_UADDR)

Bit	Mnemonic	Description	Comments
31:0	SAUADR	Security Association Upper Base Address	

18.4.32 PE Gather Ring Base Upper Address (CRYP0_PE_GATH_RING_UADDR)

This register allows the Host to specify the upper part of the location of the Gather descriptor ring in external memory space. The gather descriptor particles are fetched from: CRYP0_PE_GATH_RING_UADDR||CRYP0_PE_GATH.

Reset: 0x0000 0000

Figure 18-32. PE Gather Base Upper Address (CRYP0_PE_GATH_RING_BASE_UADDR)

Bit	Mnemonic	Description	Comments
31:0	GDRUADR	Gather Descriptor Ring Upper Base Address	

18.4.33 PE Scatter Ring Upper Address (CRYP0_SCAT_RING _UADDR)

This register allows the Host to specify the upper part of the location of the Scatter descriptor ring in external memory space. The scatter descriptor particles are fetched from: CRYP0_SCAT_RING_UADDR||CRYP0_PE_SCAT.

Reset: 0x0000 0000

Figure 18-33. PE Scatter Upper Address (CRYP0_SCAT_RING _UADDR)

Bit	Mnemonic	Description	Comments
31:0	SRUADR	Scatter Ring Upper Base Address	

18.4.34 PE Particle Descriptor Source Address Register (CRYP0_PE_PR_SCA)

This register contains the particle descriptor source address. Although this is a read/write register, in normal operation, the processor does not read or write it. Writing is not recommended since it can interfere with correct gather processing. The register should be used for debug only.

Reset: 0x0000 0000

Figure 18-34. PE Particle Descriptor Source Address Register (CRYP0_PE_PR_SCA)

Bit	Mnemonic	Description	Comments
31:0	SPS	Particle Address Starting address of the Gather Particle	Set by the processor in the External Particle Ring.

18.4.35 PE Particle Descriptor Source Control Register (CRYP0_PE_PR_SCC)

This register contains the particle descriptor source control. Although this is a read/write register, in normal operation, the processor does not read or write it. Writing is not recommended since it can interfere with correct gather processing. The register should be used for debug only.

Reset: 0x0000 0002

Figure 18-35. PE Particle Descriptor Source Control Register (CRYP0_PE_PR_SCC)

Bit	Mnemonic	Description	Comments
31:16	PS	Particle Size Size of the Scatter Particle	Set by the processor in the External Particle Ring. This has to be a value in the range of 4 to 65532 bytes in multiples of four bytes.
15:2		Reserved	
1	PEDONE	0 Security function has not finished processing the particle descriptor 1 Security function has finished processing the particle descriptor and has returned ownership to the processor	Written by the security function. Read by the processor. In conjunction with HDRDY (Bit 0): Bits 1 and 0, 00 = Unassigned. Security function ignores. 01 = Processor has populated particle data and particle descriptor. Security function now owns. 10 = Security function done. Ownership returned to the processor. 11 = Reserved
0	HDRDY	Host Ready 0 The processor has not populated the particle descriptor 1 The processor has populated the particle descriptor	Written by the processor. Read by the security function. In conjunction with PEDONE (Bit 1): Bits 1 and 0, 00 = Unassigned. Security function ignores. 01 = Processor has populated particle data and particle descriptor. Security function now owns. 10 = Security function done. Ownership returned to the processor. 11 = Reserved

18.4.36 PE Particle Destination Address Register (CRYP0_PE_PR_DTA)

This register contains starting address of the Scatter Particle, as set by the processor in the External Particle Ring. Although this is a read/write register, in normal operation, the processor does not read or write it. Writing is not recommended since it can interfere with correct gather processing. The register should be used for debug only.

Reset: 0x0000 0000

Figure 18-36. PE Particle Destination Address Register (CRYP0_PE_PR_DTA)

Bit	Mnemonic	Description	Comments
31:0	SPS	Particle Address	Set by the processor in the External Particle Ring.

18.4.37 PE Particle Destination Control Register (CRYP0_PE_PR_DTC)

This register contains particle destination control information.

Reset: 0x0000 0002

Figure 18-37. PE Particle Destination Control Register (CRYP0_PE_PR_DTC)

Bit	Mnemonic	Description	Comments
31:2		Reserved	There is no particle size field in this register. The size of the Scatter Particle is programmable (but fixed and the same for all packets during packet processing). The processor can set by this size (4 to 65532 bytes) in the Particle Ring Configuration Register (CRYP0_PE_PT_CFG).
1	PEDONE	Security function done 0 Security function has not finished processing the particle descriptor 1 Security function has finished processing the particle descriptor and has returned ownership to the processor	Written by the security function. Read by the processor. In conjunction with HDRDY (Bit 0): Bits 1 and 0, 00 = Unassigned. Security function ignores. 01 = Processor has populated particle data and particle descriptor. Security function now owns. 10 = Security function done. Ownership returned to the processor. 11 = Reserved
0	HDRDY	Host Ready 0 The processor has not populated the particle descriptor 1 The processor has populated the particle descriptor	Written by the processor. Read by the security function. In conjunction with PEDONE (Bit 1): Bits 1 and 0, 00 = Unassigned. Security function ignores. 01 = Processor has populated particle data and particle descriptor. Security function now owns. 10 = Security function done. Ownership returned to the processor. 11 = Reserved

18.4.38 SA Command 0 Register (CRYP0_SA_CMD_0)

The Security Association (SA) record, along with the packet descriptor, provides the PE all of the necessary information to process an operation. The SA-record contains information which is either static for the lifetime of the association or is dynamically updated by the PE. Any control information which must be modified by the processor for each operation is contained in the descriptor.

This register contains the major control bits for defining the cryptographic operation. It is Write-only and always returns zero on a read by the processor.

Reset: 0x0000 0000

Figure 18-38. SA Command 0 Register (CRYP0_SA_CMD_0)

Bit	Mnemonic	Description	Comments
31	OSCAT	Output Scatter Controls whether output packet data is scattered 0 Output packet data is contiguous in memory 1 Output packet data must be scattered to memory	
30	IGATH	Input Gather Controls whether input packet data is gathered 0 Input packet data is contiguous in memory 1 Input packet data must be gathered from memory	
29	SHSTT	Save Hash State Controls saving the hash state after completion of an MD5, SHA-1, SHA-2, HMAC, or KASUMI f9 operation. 0 Do not save the hash state to the State Record 1 Save the hash state to the State Record	The hash state includes the interim hash digest and the hash byte count.
28	SIV	Save IV Controls saving the IV to the State Record after completion of a DES/Triple-DES or AES operation. 0 Do not save the IV to the State Record 1 Save the IV to the State Record	This bit must be cleared for DES ECB mode or AES ECB mode, when no IV is used.
27:26	LHSTT	Load Hash State Controls Hash loading for the MD5, SHA-1, SHA-2, and KASUMI f9 algorithms. 00 From SA or State (digest only, Hash Byte Count = 0x40) 01 Reserved 10 From state (read Saved Inner Hash Digest and Saved Hash Byte Count) 11 No load (use the Hash algorithm defined constants for the initial Hash. Hash Byte Count = 0x00)	

Figure 18-38. SA Command 0 Register (CRYP0_SA_CMD_0) (Continued)

Bit	Mnemonic	Description	Comments
25:24	IVLD	IV Loading Controls IV loading for the AES, DES, Triple-DES, and KASUIM f8 cryptographic algorithms. For cryptographic algorithms that do not require an IV, these bits are ignored. 00 Reuse (no load) - not applicable for inbound data 01 From input buffer 10 From State record 11 From internal PRNG	
23:20	DLEN	Digest Length. Length of Hash Digest, in words. 0000 No hash digest output 0001 1 word 0010 2 words 0011 3 words 0100 4 words 0101 – 1111 Reserved	Only applicable for IPSec ESP operations that use AES-CBC-MAC operations.
19	HP	Header Processing Controls how the security function performs header processing for this SA-record flow. This bit is only applicable for IPsec Protocol operations. There is no Header Processing for Basic Operations, SSL, TLS and SRTP protocol operation. However, for SSL Hash and TLS Hash operations the Header must be supplied to the PE. The same is applicable for SRTP Encryption-Hash operations. **IPSEC Outbound** (3:0 = 0000)—Determines whether the security function performs IPsec header insertion. 0 Processor subsystem is expected to insert either an AH or ESP header into the correct location within a packet. The security function inserts the ICV into the provided hole in the AH header. 1 Security function inserts an AH or ESP header in the proper location for an outbound packet, including the SPI and Sequence Counter. **IPSEC Inbound** (Bits 3:0 = 1000)—Determines whether the security function performs IPsec header verification. 0 Processor subsystem is expected to verify the AH or ESP header. 1 Security function verifies the AH or ESP header in the inbound packet, including the Sequence Number verification and Sequence mask processing	Fragment Headers processing is not supported by the security function. Any Fragment Header passed into the PE is ignored. So the Host *must* make sure that fragmentation headers are stripped before entering the packet in the security function. Should the Fragment Header be present in the packet, it is treated as an extension header (that is, it is ignored as being a Fragment Header, but not skipped over and the previous header's Next Header field is not changed to be the Next Header field in the Fragment Header). The length field of this header is fixed at 64 bits. For an IPsec IPv6 inbound packet any order of Extension Headers is allowed. The security function uses the following rules for placing an AH Header in an IPv6 outbound packet: 1. The AH Header is always inserted before the payload data where payload is defined as either the protocol data or the start of a tunneled packet. 2. If there is one Destination Header the AH Header will be inserted between the Destination Header and the payload. 3. If there are multiple Destination Headers before the payload then the AH header is inserted before the second Destination Header. A possible third Destination Header (although not allowed by the RFC 2402) is seen as payload.
18	EPAD	Extended Pad Extends the number of Pad options for the security function 0 Basic Pad modes 1 Extended Pad modes required for SSL and TLS protocol operations	See CPAD.
17	PSC	Pad Stream Ciphers Controls padding for Basic operations that use Stream Ciphers. 0 Disable padding 1 Enable padding	Protocols operations that use Stream Ciphers are always padded according to the protocol requirements.

Figure 18-38. SA Command 0 Register (CRYP0_SA_CMD_0) (Continued)

Bit	Mnemonic	Description	Comments
16		Reserved	
15:12	HALGO	Select the Hash Algorithm 0000 MD5 0001 SHA-1 0010 SHA2-224 0011 SHA2-256 0100 SHA2-384 0101 SHA2-512 0110 – 0111 Reserved 1000 AES-XCBC-MAC-128 bit key 1001 KASUMI 19 1010 – 1011 Reserved 1100 GHASH (when AES-CTR = CCM) 1101 GMAC 1110 CBC-MAC (when AES-CTR = CCM) 1111 Null	Not all hash algorithms are allowed for protocol operations.
11:8	CALGO	Crypto Algorithm Selects the Symmetric Encryption/Decryption algorithm. Not all algorithms are allowed for protocol operations. 0000 DES 0001 Triple-DES 0010 ARC4 0011 AES 0100 KASUMI 0101 – 1110 Reserved 1111 Null	
7:6	CPAD	Crypto Pad Selects the method of cryptographic padding. If EPAD = 0: 00 IPsec 01 PKCS#7 10 Constant Pad 11 Zero If EPAD = 1: 00 Reserved 01 TLS 10 Constant SSL 11 Reserved	Set padding type IPSec (EPAD = 0, CPAD = 00) for SGT L3. The padding bytes appended to the plain text are numbered 1, 2, 3, and so on. IPSec padding with the pad field set to all zeros is not supported.
5:4	OGRP	Operation Group: These two bits select one of four groupings of operations: 00 Basic Operations 01 Protocol operation group 10 Extended operation group 11 Reserved	See tabxref1 and tabxref2.
3	IO	Inbound/Outbound processing 0 Outbound 1 Inbound	For example, the encryption and compression operations are considered Outbound, and decryption and decompression are considered Inbound.
2:0	OCODE	OpCode These three bits select a specific sub-operation from within the Operation Group.	See *Table 18-14* and *Table 18-15* below.

Table 18-14. Basic Operation Decoding

Outbound				Inbound			
Operation Group	**In/Out**	**OpCode**	**Description**	**Operation Group**	**In/Out**	**OpCode**	**Description**
00	0	000	Encryption	00	1	000	Decryption
00	0	001	Encryption-Hash	00	1	001	Hash-Decryption
00	0	010	Reserved (Compress)	00	1	010	Reserved (Decompress)
00	0	011	Hash	00	1	011	Hash
00	0	100	Reserved (Hash-Encryption)	00	1	100	Reserved (Decryption-Hash)
00	0	101–111	Reserved	00	1	101–111	Reserved

Table 18-15. Protocol Operation Decoding

Outbound				Inbound			
Operation Group	**In/Out**	**OpCode**	**Description**	**Operation Group**	**In/Out**	**OpCode**	**Description**
01	0	000	ESP Outbound	01	1	000	ESP Inbound
01	0	001	AH Outbound	01	1	001	AH Inbound
01	0	010	Reserved for Ipcomp	01	1	010	Reserved for Ipcomp
01	0	011	Reserved for MPPE	01	1	011	Reserved for MPPE
01	0	100	SSL Outbound (See **Note**)	01	1	100	SSL Inbound (See **Note**)
01	0	101	TLS Outbound (See **Note**)	01	1	101	TLS Inbound (See **Note**)
01	0	110	Reserved for WTLS	01	1	110	Reserved for WTLS
01	0	111	SRTP Outbound (See **Note**)	01	1	111	SRTP Inbound (See **Note**)
Note: This is not a full protocol operation. For SSL/TLS and SRTP no Headers processing is performed in hardware.							

Table 18-16. Extended Protocol Operations

Outbound				Inbound			
Operation Group	**In/Out**	**OpCode**	**Description**	**Operation Group**	**In/Out**	**OpCode**	**Description**
11	0	000	Reserved	11	1	000	Reserved
11	0	001	DTLS Outbound	11	1	001	DTLS Inbound
11	0	010	MACSEC Outbound	11	1	010	MACSEC Inbound
11	0	011	Reserved	11	1	011	Reserved
11	0	100	SSL Outbound	11	1	100	SSL Inbound
11	0	101	TLS v1.0 Outbound	11	1	101	TLS v1.0 Inbound
11	0	110	TLS v1.1 Outbound	11	1	110	TLS v1.1 Inbound
11	0	111	Reserved	11	1	111	Reserved

18.4.39 SA Command 1 Register (CRYP0_SA_CMD_1)

This register contains the minor control bits for defining a cryptographic operation. See the companion register CRYP0_SA_CMD_0.

It is Write-only and always return zero on a read by the processor.

Reset: 0x0000 0000

Figure 18-39. SA Command 1 Register (CRYP0_SA_CMD_1)

Bit	Mnemonic	Description	Comments
31	CMODE	Cryptographic Mode See description of bits 9:8.	Used in combination with bits 9:8 in this register.
30	SARC4S	Save ARC4 State Controls whether the PE writes the ARC4 State data back out to the SA-record. 0 Do not save the ARC4 state 1 Save the ARC4 State	This bit is normally set for Stateful ARC4 and not set for Stateless.
29	ARC4SS	ARC4 Stateless/Stateful Controls whether the ARC4 engine is running in the Stateless or Stateful mode. 0 Stateless 1 Stateful	**Stateless:** Each packet is processed with a newly initialized ARC4 key taken from the Key field of the SA-record. In this mode, the state information from the SA is never read. **Stateful:** When bit 3, Init ARC4, of the CRYP0_PE_CTLST register is 1, the ARC4 algorithm initializes using the Key specified in the SA-record. When the Init ARC4 bit is 0, the ARC4 context is read from the ARC4 State field of the SA-record and the i/j Pointer field of the SA. The encryption/decryption processing continues from this previous algorithm state.
28:24	ARC4KL AESKL	**28:24** ARC4 Key Length Selects the key length, in bytes, for an ARC4 and AES operations only. Valid settings range from 0x00001 to 0x10000 corresponding to 8–128 bits, in steps of 8 bits. For SSL and TLS operations specifically key sizes of 5B and 16B are used. **26:24** AES Key Length: Selects the size of the key data used for AES operations only. The key length changes in increments of 64 bits: 00000 – 00001 Reserved 00010 128 bits 00011 192 bits 00100 256 bits 00101– 11111 Reserved	

Figure 18-39. SA Command 1 Register (CRYP0_SA_CMD_1) (Continued)

Bit	Mnemonic	Description	Comments
23:16	HCO	Hash/Encryption Offset Specifies the offset, in words, between the hash data and the encryption/decryption data for Basic operations. In the case of Outbound operations, the data to be hashed is assumed to come first, with an offset to the beginning of encrypted data. For Inbound operations, the data to be hashed is assumed to come first, with an offset to the beginning of data to decrypted. For SRTP there is always an offset. For other operations these bits are not used (a default value is applied by the PE). If an IV is loaded through the input buffer, the Hash/Encryption Offset must include the IV. In the case of DES and Triple-DES and AES-CTR operations, the offset is 0x02 words. In the case of AES-CBC and AES-ICM, the offset is 0x04 words. For MACSec protocol operations, these bits specify the confidentiality offset in words.	
15		Reserved	
14	SAREV	SA Revision Specifies the revision level of the SA structure. 0 Revision 1 1 Revision 2	
13	BOFFSET	Byte offset 0 HCOffset is defined in 32-bit words 1 HCOffset is define din 8-bit words	This bit only used for MACSec protocol operations.
12	HMAC	HMAC/IP Options Muting Control This bit has two functions: **Basic operations that include hashing:** Enables the HMAC processing, which calls for an outer hash operation to occur. 0 Standard hash processing 1 HMAC processing **IPsec AH operations:** Controls Mutable-bit processing on IPv4 options and IPv6 extension headers. 0 Enables Mutable-bit processing on options and extension headers 1 Disables mutable-bit processing	For HMAC operations, pre-computed hash digests must be used and written to the inner and outer digest fields in the SA-Record. If the Hash Final bit in the descriptor is 0 this bit is don't care (overruled) and a standard hash is processed. In case the Hash Final bit in the descriptor is 1 and standard hash processing is selected, the data is hashed and the required hash padding is appended. In case the Hash Final bit in the descriptor is 1 and HMAC is selected, an outer hash operation is applied. Mutable-bit processing for the IP header itself is specified in the MBP field. This bit is don't care (overruled) for ESP, SSL, TLS and SRTP operations. For IPv6 IPsec AH operations, Inbound and Outbound, a Destination Option header after the AH header is never muted.
11:10	CFDBK	Cryptographic Feedback Mode Selects the cryptographic feedback mode for OFB and CFB modes. 00 64-bit OFB/CFB for DES/Triple-DES 01 8-bit CFB for DES/Triple-DES/AES 10 1-bit CFB for DES/Triple-DES/AES 11 128-bit CFB for AES	

Figure 18-39. SA Command 1 Register (CRYP0_SA_CMD_1) (Continued)

Bit	Mnemonic	Description	Comments
9:8	CMODE	Cryptographic Mode Selects the encryption mode for Basic Encryption operations. 000 DES/Triple DES/AES ECB (Electronic Code Book) or KASUMI 001 DES/Triple DES/AES CBC (Cipher Block Chaining) 010 DES/Triple DES/AES CBC OFB (Output Feed Back) 011 DES/Triple DES/AES CBC CFB (Cipher Feed Back) 100 AES-CTR (32-bit Counter, Counter Mode for IPsec) or KASUMI f8 101 AES-ICM (16-bit Counter, Integer Counter Mode for SRTP) 110 – 111 Reserved.	Bits 31 and 9:8 denote the Cryptographic mode. See *Table 18-17*.
7	ESN	Extended Sequence Numbers (ESN) 0 32-bit Sequence Number 1 64-bit Sequence Number	Only applicable for IPSec operations.
6	SNMASK	Sequence number mask 0 128-bit Sequence number mask 1 64-bit Sequence number mask	Only applicable for IPSec protocol operations.
5	MBP	IP Header Mutable Bit Handling Controls Mutable-bit zeroing on the IP header as specified in RFC 2402 and RFC 2460, the AH Protocol RFCs to allow the security processor to determine where the AH header is located. 0 Enable Mutable-bit processing 1 Disable Mutable-bit processing	The processor should replace predictable fields before offering the packet to the security function. After the packet is returned, the processor should put the initial values back again. For example, the security processor does not automatically adjust the dest_addr field of the base header to the final destination address according to the routing entries. Similarly, the Routing Header extension (type 0) is also not adjusted. This bit specifies Mutable-bit processing only for the IP header. HMAC in this register controls whether the IPv4 options and IPv6 extension headers are also muted. This bit is ignored for all non-AH operations. For Hash operations that do not require muting, this bit should always be set to 1.
4	IP4IP6	IPv4/IPv6 Selects the IP protocol version for this packet flow. 0 IPv4 1 IPv6	This is required for Mutable-bit processing (see MBP above) and in the AH protocol to allow the security function to determine where the AH header is located.
3	CRYP0	Copy Inbound Pad to Output Controls whether the PE transfers padding information from the input buffer to the output buffer. 0 Pad is not copied to the output buffer. 1 Pad is copied to the output buffer.	This bit is only applicable for the following operations and the pad is defined as all data: IPsec ESP Inbound: between the payload and the ICV. SSL Inbound: after the payload (this includes the MAC). TSL Inbound: after the payload (this includes the MAC). Basic Decryption: after the payload. Basic Hash-Decryption: after the payload.

Figure 18-39. SA Command 1 Register (CRYP0_SA_CMD_1) (Continued)

Bit	Mnemonic	Description	Comments
2	CRYPLO	Copy Payload to Output Controls whether the PE transfers payload information from the input buffer to the output buffer. 0 Payload is not copied to the output buffer 1 Payload is copied to the output buffer	This bit is only applicable for the following operations and the payload is: IPsec AH Inbound: defined as all data after the AH. Except in case a second Destination Option Header directly follows the AH it is defined as part of the Header (refer to RFC 2402). IPsec AH Outbound: defined the AH Header and the payload, excluding the second Destination Header. Basic Hash: the entire data to be hashed. For all other operations the payload is always copied from the input buffer to the output buffer.
1	CRYPHO	Copy Header to Output Controls whether the PE transfers header information from the input buffer to the output buffer. 0 Header is not copied to the output buffer 1 Header is copied to the output buffer	This bit is only applicable for the following operations and the header is defined as the: IPsec ESP Inbound: hash/encryption offset data (authenticated only). IPsec AH Outbound: data from the start of a packet to the beginning of the ICV field in the AH header and, if available, includes the Destination Header after the ICV. IPsec AH Inbound: data from the start of a packet to the end of the ICV field and, if available, includes the Destination Header after the ICV. If bits 1 and 2 are both 0, then only the ICV will be returned. SSL Outbound and Inbound: Sequence Number, Type and Length fields. This bit must be set for SSL-Null encryption. TLS Outbound and Inbound: Sequence Number, Type and Length fields. This bit must be set for TLS-Null encryption. SRTP Outbound and Inbound: Bypass Offset, the data from the start of the packet to the start of the payload. In general this bit will be set to 0 since no fields in the header need to be updated. Basic Encryption-Hash and Basic Hash-Decryption: Hash/Cryption Offset data (authenticated only).
0		Reserved	

Table 18-17. Combined Algorithm Modes

CRYP0_SA_CMD_0		CRYP0_SA_CMD_1	
Cipher[11:8]	**Hash[15:12]**	**CMode[31,9:8]**	**Description**
0011	1100	100	Galois Counter Mode (GCM)
0011	1101	100	Galois Message Authentication Code (GMAC)
0011	1110	100	Counter with Cipher Block Chain Mode (CCM) 32-bit counter
0011	1110	101	Counter with Cipher Block Chain Mode (CCM) 16-bit counter

18.4.40 SA Key x Registers (CRYP0_SA_KEYx)

There are eight KEY registers (where x = 0–7). The first four registers, CRYP0_SA_KEY0:3, contain the key to the KASUMI and KASUMI f8 algorithms.

Reset: 0x0000 0000

Figure 18-40. SA Key x Registers (CRYP0_SA_KEYx)

Bit	Mnemonic	Description	Comments
31:0		DES/Triple-DES/ARC4/AES Key Key bits 31:0 are in CRYP0_SA_KEY0 Key bits 63:32 are in CRYP0_SA_KEY1	For a Single-DES key, only KEY0 is used. For a Triple-DES key, the first three pairs of registers (KEY0, KEY1 and KEY2) are used. For ARC4, KASUMI, and KASUMI f8 the first four registers (KEY0 – KEY3) are used. For an AES: a. 128-bit key uses KEY0 and KEY1 b. 192-bit key uses KEY0 – KEY5 c. 256-bit key uses KEY0 – KEY7

18.4.41 SA Inner Hash Digest x Registers (CRYP0_SA_IH_Dx)

These sixteen registers (x = 0–15) contain the 512-bit inner hash digest value. For operations that make use of the precomputed inner hash digest, the value is written into these registers.

For operations that generate an interim hash value, these registers contain the interim hash.

See *SA Inner Hash x (mirror of CRYP0_SA_IH_Dx) Registers (CRYP0_SA_IH_x)* on page 381

The first four registers, CRYP0_SA_IH_D0:3, contain the integrity key to the KASUMI f9 algorithm.

Reset: 0x0000 0000

Figure 18-41. SA Inner Hash Digest x Registers (CRYP0_SA_IH_Dx)

Bit	Mnemonic	Description	Comments
31:0		Inner Hash Digest Bits 31:0 are in CRYP0_SA_IH_D0. Bits 63:32 are in CRYP0_SA_IH_D1. Bits 95:64 are in CRYP0_SA_IH_D2. Bits 127:96 are in CRYP0_SA_IH_D3 Bits 159:128 are in CRYP0_SA_IH_D4. Bits 191:160 are in CRYP0_SA_IH_D5. Bits 223:192 are in CRYP0_SA_IH_D6. Bits 255:224 are in CRYP0_SA_IH_D7. Bits 287:256 are in CRYP0_SA_IH_D8. Bits 319:288 are in CRYP0_SA_IH_D9. Bits 351:320 are in CRYP0_SA_IH_D10. Bits 383:352 are in CRYP0_SA_IH_D11. Bits 415:384 are in CRYP0_SA_IH_D12. Bits 447:416 are in CRYP0_SA_IH_D13. Bits 479:448 are in CRYP0_SA_IH_D14. Bits 511:480 are in CRYP0_SA_IH_D15.	For Basic Hash (MD5, SHA-1, and SHA-2) these registers are for both starting hash and interim or final hash. For IPSec and TLS and SSL hash operations (MD5, SHA-1, and SHA-2) or HMAC these registers contain the pre-computed inner hash. For KASUMI f9 the first four registers (CRYP0_SA_IH_D0:3) contain the integrity key.

18.4.42 SA Outer Digest x Registers (CRYP0_SA_OH_Dx)

When *writing*, these 16 registers (x = 0–15) contain the 512-bit pre-computed outer hash digest for IPsec operations with authentication, or basic HMAC operations with Load Hash from SA set. They are used for writing the starting or interim outer hash digest, and for HMAC processing only.

When *reading*, these 16 registers contain the MAC hash result used for SSL and TLS and has the same functionality as the ICV register for IPsec.

The first three registers, CRYP0_SA_OH_D0:2, contain the variables to the KASUMI f9 algorithm.

Reset: 0x0000 0000

Figure 18-42. SA Outer Hash Digest x Registers (CRYP0_SA_OH_Dx)

Bit	Mnemonic	Description	Comments
31:0		Outer Hash Digest/MAC Hash Result Bits 31:0 are in CRYP0_SA_OH_D0. Bits 63:32 are in CRYP0_SA_OH_D1. Bits 95:64 are in CRYP0_SA_OH_D2. Bits 127:96 are in CRYP0_SA_OH_D3 Bits 159:128 are in CRYP0_SA_OH_D4 Bits 191:160 are in CRYP0_SA_OH_D5 Bits 223:192 are in CRYP0_SA_OH_D6 Bits 255:224 are in CRYP0_SA_OH_D7 Bits 287:256 are in CRYP0_SA_OH_D8 Bits 319:288 are in CRYP0_SA_OH_D9 Bits 351:320 are in CRYP0_SA_OH_D10 Bits 383:352 are in CRYP0_SA_OH_D11 Bits 415:384 are in CRYP0_SA_OH_D12 Bits 447:416 are in CRYP0_SA_OH_D13 Bits 479:448 are in CRYP0_SA_OH_D14 Bits 511:480 are in CRYP0_SA_OH_D15	When *writing*; For MD5, only the first four registers are used. For SHA-1, all five registers are used. For SHA-2, the number of registers used depends on the size of the hash digest. When *reading*: For MD5 the MAC is 16 bytes. For SHA-1 the MAC is 20 bytes. For SHA-2, the size of the MAC depends on the size of the hash digest. For KASUMI f9, CRYP0_SA_OH_D0:2 contain variables to the algorithm. CRYP0_SA_OH_D0 is a 32-bit counter variable, COUNT. CRYP0_SA_OH_D1 is a 32-bit random input variable, FRESH. CRYP0_SA_OH_D2 is a 1-bit direction input variable, DIRECTION.

18.4.43 SA IPsec SPI Register (CRYP0_SA_SPI)

This register has two functions: For IPsec operations, it is written with the SPI (Security Parameters Index) associated with the inbound or outbound flow. In addition, it is the trigger register for indicating that the key and hash digests have been written.

This register *must* be written for all Direct Host DMA packet operations, even if the SPI is not used (any value may be written for non-IPSEC operations). This register should be written "in sequence" as the entire SA-record is written. Therefore, it must be written after the key and hash digests, but it can be written before the Sequence Number and Mask registers.

Reset: 0x0000 0000

Figure 18-43. SA IPsec SPI Register (CRYP0_SA_SPI)

Bit	Mnemonic	Description	Comments
31:0		SA IPsec SPI Used for IPsec ESP and AH operations to specify the Security Parameters Index (SPI) value that is to be placed in the ESP or AH header.	The description applies only to the Direct Host DMA mode. For Autonomous Ring mode and Target Command mode, the security function extracts the SPI from of the SA-Record. There is no need to read back this value at the end of an operation, since the security function will not change it.

18.4.44 SA IPsec Sequence Number Registers 0 and 1 (CRYP0_SA_SEQx)

These registers are used for IPsec ESP and AH operations to specify the anti-replay sequence number value that is to be placed in the ESP or AH header for outbound operations, or to be checked against for inbound packets. The security function manages this counter value for both inbound and outbound operations.

Reset: 0x0000 0000

Figure 18-44. SA IPsec Sequence Number Registers 0 and 1 (CRYP0_SA_SEQx)

Bit	Mnemonic	Description	Comments
31:0		Sequence Number **Outbound:** The processor writes the counter value stored in the SA-record into this register when starting an IPsec operation. The security function automatically increments the count. Upon completion, the processor can read this value and write it to the SA-record. **Inbound:** The processor writes the counter value stored in the SA-record into this register when starting an IPsec operation. The security function automatically performs the specified inbound processing (per RFC2402 and RFC2406) as it processes the packet. As a result, the expected count value might or might not be updated during processing. Upon completion, the processor reads back this value and writes it to the SA-record.	The description only applies to the Direct Host DMA mode. For Autonomous Ring mode and Target Command mode, the sequence number is updated by the PE.

18.4.45 SA IPsec Sequence Number Mask Registers (CRYP0_SA_SEQMKx)

These four registers (x = 0–4) are used for IPsec ESP and AH operations to specify the anti-replay sequence number mask value for inbound operations. The security function manages this counter value automatically.

Reset: 0x0000 0000

Figure 18-45. SA IPsec Sequence Number Mask Registers (CRYP0_SA_SEQMKx)

Bit	Mnemonic	Description	Comments
31:0		Sequence Number Mask Bits 31:0 are in CRYP0_SA_SEQMKL Bits 63:32 are in CRYP0_SA_SEQMKH **Outbound:** Not used. **Inbound:** The processor writes the counter value stored in the SA-record into this register upon starting an IPsec operation. The security function automatically performs the specified inbound processing (per RFC2402 and RFC2406) as it processes the packet. As a result, the new mask value might or might not be updated during processing. Upon completion, the processor reads this value and writes it to the SA-record.	The description only applies to the Direct Host DMA mode. For Autonomous Ring mode and Target Command mode, the security function extracts the Sequence Number Mask from of the SA-Record.

18.4.46 SA Pointer Register (CRYP0_SA_PNTR)

This register contains the address pointer to the state record.

Reset: 0x0000 0000

Figure 18-46. SA Pointer Register (CRYP0_SA_PNTR)

Bit	Mnemonic	Description	Comments
31:0		Pointer	Direct Host DMA mode only.

18.4.47 SA ARC4 i and j Pointer Register (CRYP0_SA_ARC4IJ)

When starting a new ARC4 operation this register contains the initialization value (0). After processing the ARC4 algorithm, it contains the latest status of the ARC4_I_J_PNTR.

Reset: 0x0000 0000

Figure 18-47. SA ARC4 i and J Pointer Register (CRYP0_SA_ARC4IJ)

Bit	Mnemonic	Description	Comments
31:16		Reserved	
15:8	JPNT	j Pointer Pointer into s-box array for swapping bytes with i pointer.	The description applies only to the Direct Host DMA mode. For Autonomous Ring mode and Target Command mode the security function extracts the pointer from of the SA-Record.
7:0	IPNT	i Pointer Pointer into s-box array for swapping bytes with j pointer	The description applies only to the Direct Host DMA mode. For Autonomous Ring mode and Target Command mode the security function extracts the pointer from of the SA-Record.

18.4.48 SA ARC4 State Address Pointer Register (CRYP0_SA_ARC4SB)

This register contains the pointer to externally located ARC4 SBOX data.

Reset: 0x0000 0000

Figure 18-48. SA ARC4 State Address Pointer Register (CRYP0_SA_ARC4SB)

Bit	Mnemonic	Description	Comments
31:0		ARC4 SBOX Pointer Register	Direct Host DMA mode only.

18.4.49 SA Initialization Vector Registers (CRYP0_SA_IV_x)

These four registers contain the 128-bit initialization vector.

KASUMI f9 uses CRYP0_SA_IV_0 to store the COUNT variable and CRYP0_SA_IV_1 to store the BEARER (in bits 7:3) and DIRECTION (in bit 2) variables.

Reset: 0x0000 0000

Figure 18-49. SA Initialization Vector Registers (CRYP0_SA_IV_x)

Bit	Mnemonic	Description	Comments
31:0		Initilization Vector (IV) Bits 31:0 are in CRYP0_SA_IV_0 Bits 63:32 are in CRYP0_SA_IV_1 Bits 95:64 are in CRYP0_SA_IV_2 Bits 127:96 are in CRYP0_SA_IV_3 These registers are used both for entering a starting IV state as well as for reading the interim or final IV. For operations that use cryptographic mode AES-ICM, CRYP0_SA_CMD_1[31, 9:8] is 100, CRYP0_SA_IV_3[31:0] are the block counter. For operations that use crypto mode AES-CTR, CRYP0_SA_CMD_1[31,9:8] is 101, CRYP0_SA_IV_3[31:0] are the block counter and CRYP0_SA_IV_0[31:0] are the Nonce.	Direct Host DMA mode only. For IPsec *outbound* operations, it is recommended that the automatic IV insertion mode be used, which means that these registers are not needed. For IPsec *inbound* operations, the IV is extracted from the header of the packet. For the KASUMI f8, CRYPO_SA_IV_0:1 contain variables for the algorithm. CRYPO_SA_IV_0 is a 32-bit counter variable COUNT. CRYPO_SA_IV_1 bit 2 is DIRECTION, and bits 7:3 are BEARER, all other are Reserved.

18.4.50 SA Hash Byte Count Register (CRYP0_SA_HASH_Bx)

This register is used both for entering a starting hash byte count, as well as for reading the interim or final byte count.

For some hash operations, this register is ignored and the byte count is internally set to 64 to indicate that the first 64-byte hash block has been processed using a pre-computed hash state. These operations include:

- All IPsec OpCodes that use authentication (pre-computed inner and outer hash digests are loaded from SA words 10–19)
- Basic Operations with Load Hash from SA specified. For Basic Hash with no HMAC, a pre-computed digest is loaded from SA words 10–14. For Basic HMAC, the inner and outer digests are loaded from SA words 10–19.

Note: IPsec operations can not be suspended in mid-packet and resumed later, and so do not use this register.

Reset: 0x0000 0000

Figure 18-50. SA Hash Byte Count Register (CRYP0_SA_HASH_Bx)

Bit	Mnemonic	Description	Comments
31:0		Hash Byte Count	Direct Host DMA mode only.

18.4.51 SA Inner Hash x (mirror of CRYP0_SA_IH_Dx) Registers (CRYP0_SA_IH_x)

See *SA Inner Hash Digest x Registers (CRYP0_SA_IH_Dx)* on page 375.

Reset: 0x0000 0000

Figure 18-51. SA Inner Hash x (mirror of CRYP0_SA_IH_Dx) Registers (CRYP0_SA_IH_x)

Bit	Mnemonic	Description	Comments
31:0		Mirror of CRYP0_SA_IH_Dx	

18.4.52 SA ICV x—HMAC Result Registers (CRYP0_SA_ICV_x)

These registers are used to store the ICV value for IPsec, SSL and TSL operations. For outbound operations they contain the hash result. For inbound operations they contain the ICV that will be checked against the hash result.

For ESP and AH inbound operations, the ICV registers are typically not accessed by software. The ICV value is normally calculated internally and automatically compared against the ICV contained in the packet. The authentication result is then provided in the result descriptor.

For ESP outbound operations, the ICV registers are typically not used, as the ICV value is inserted in the data output buffer directly after the payload and is read out of the data output FIFO along with the processed packet.

For an AH outbound operation, the ICV registers are typically not used, as the ICV value is inserted in the data output buffer directly in front of the payload and is read out of the data output FIFO along with the processed packet. When the packet size is larger than the buffer size, the processed packet is read from the data output buffer and then the ICV is updated in memory.

Reset: 0x0000 0000

Figure 18-52. SA ICV x—HMAC Result Registers (CRYP0_SA_ICV_x)

Bit	Mnemonic	Description	Comments
31:0		HMAC Result Bits 31:0 are in CRYP0_SA_ICV_0 Bits 63:32 are in CRYP0_SA_ICV_1 Bits 95:64 are in CRYP0_SA_ICV_2 Bits 127:96 are in CRYP0_SA_ICV_3 Bits 159:128 are in CRYP0_SA_ICV_4 Contain the 160-bit ICV for IPsec authentication operations. This is the computed ICV value (as opposed to the ICV extracted from an inbound packet). The ICV contained here differs from the hash digest registers above in that for a SHA-1 or SHA-2 hash, the ICV field is endian-swapped from the hash digest field. This is consistent with the IPsec packet format standards.	

18.4.53 TRNG Output Register (CRYP0_TRNG_DATA)

This register contains a word of random data. The processor is responsible for monitoring the True Random Number Generator (TRNG) status register to make sure that the data presented is ready.

Reset: 0x0000 0000

Figure 18-53. TRNG Output Register (CRYP0_TRNG_DATA)

Bit	Mnemonic	Description	Comments
31:0		TRNG Data When CRYP0_TRNG_CTRL[12:11] = 00 this register contains the 32-bit Non-Linear Mixer true random number output. When CRYP0_TRNG_CTRL[12] = 1 this register contains the 64-bit FIPS 140-2 Post-Processor random number output. The first read provides the lower 32 bits, the second read provides the higher 32 bits. When this register is read, the TRNG sets its Busy status bit and automatically begins generating a new random word. When CRYP0_TRNG_CTRL[12:11] = 11 the 64-bit FIPS 140-2 Post-Processor data input must be provided in this register. The first write must provide the lower 32 bits, the second write must provide the higher 32 bits.	See *TRNG Control Register (CRYP0_TRNG_CTRL)* on page 383.

18.4.54 TRNG Status Register (CRYP0_TRNG_STAT)

This register contains the TRNG busy indicator.

Reset: 0x0000 0001

Figure 18-54. TRNG Status Register (CRYP0_TRNG_STAT)

Bit	Mnemonic	Description	Comments
31:1		Reserved	
0	B	Busy 0 Valid random number is available in the output register 1 TRNG is busy generating the next random number	The bit is set to 1 when the TRNG Output register is read, and the TRNG starts generating the next random number.

18.4.55 TRNG Control Register (CRYP0_TRNG_CTRL)

This register is used to test the TRNG operation.

Reset: 0x0000 0400

Figure 18-55. TRNG Control Register (CRYP0_TRNG_CTRL)

Bit	Mnemonic	Description	Comments
31:13		Reserved	
12	EPP	Control the Post Processor 0 Disable the Post Processor 1 Enable the Post Processor	See *TRNG Output Register (CRYP0_TRNG_DATA)* on page 382.
11	BTRNG	Bypass TRNG 0 Normal run state 1 Enable direct access to the FIPS 140-2 Post Processor	See *TRNG Output Register (CRYP0_TRNG_DATA)* on page 382.
10	NRLFSR	Reset the Linear Feedback Shift Registers (LFSRs) 0 Reset the LFSRs within the TRNG 1 Normal run state	
9	TLFSR	Test LFSRs 0 Normal run state 1 Initiates a self-test of the LFSRs in the TRNG where the system clock replaces the ring oscillators as the LFSR source	
8	TALM	Test Alarm 0 Normal run state 1 Forces a simulated alarm state	The alarm counter increments on each system clock.
7	SC	Short Cycle 0 Normal run state 1 Shortens the state timers for simulation and testing	
6	TCNT	Clock on 0 Normal run state 1 Enables the test clock to be the system clock and allows the ring outputs to be processed in test mode	
5	DALM	Disable Alarm 0 Normal run state 1 Disables TRNG alarm reporting and does not reset the TRNG when the alarm is triggered	
4	TRNG2	Ring Oscillator #2 test 0 Disable test 1 Enable test	
3	TRNG1	Ring Oscillator #1 test 0 Disable test 1 Enable test	
2	TRUN	Enable Test Run 0 Normal run state 1 Starts the test state machine sequence	All cycles of the TRNG Finite State Machine (FSM) are tested.

Figure 18-55. TRNG Control Register (CRYP0_TRNG_CTRL) (Continued)			
Bit	**Mnemonic**	**Description**	**Comments**
1	TMODE	Enable Test Mode 0 Normal run state 1 Enables the internal TRNG registers to be written or read by the processor	Written registers: Entropy, Seed and Counter Read registers: Entropy, Seed, Counter, and LFSR
0	TRNG	Enable Test Ring Output 0 Normal run state 1 Ring oscillator output is not passed through the LFSRs in generating the random number	

18.4.56 TRNG Entropy A Register (CRYP0_TRNG_ENTA)

This register provides the read access to the Entropy A register for test and diagnostic purposes. It is accessible in Test Mode only.

Reset: 0x0000 0000

Figure 18-56. TRNG Entropy A Register (CRYP0_TRNG_ENTA)			
Bit	**Mnemonic**	**Description**	**Comments**
31:16		Reserved	
15:0	EA	Entropy A Contains the entropy output of the Ring Oscillator/ LFSR1 pair.	

18.4.57 TRNG Entropy B Register (CRYP0_TRNG_ENTB)

This register provides the read access to the Entropy B register for test and diagnostic purposes. It is accessible by the processor in Test Mode only.

Reset: 0x0000 0000

Figure 18-57. TRNG Entropy B Register (CRYP0_TRNG_ENTB)			
Bit	**Mnemonic**	**Description**	**Comments**
31:16		Reserved	
15:0	EB	Entropy B Contains the entropy output of the Ring Oscillator/ LFSR2 pair.	

18.4.58 TRNG Test Seed x Registers (CRYP0_TRNG_Xx)

These registers provide read access to the X (x = 0, 1, or 2) registers for test and diagnostic purposes. They are accessible in Test Mode only.

Reset: 0x0000 0000

Figure 18-58. TRNG Test Seed x Registers (CRYP0_TRNG_Xx)

Bit	Mnemonic	Description	Comments
31:0	RNGX	Xx Registers X0 = LFSR[31:0] X1 = LFSR[63:32]: X2 = LFSR[80:64]	X0 – 32 least significant state bits of the X LFSR. X1 – 32 middle state bits of the X LFSR. X2 – 17 most significant state bits of the X LFSR.

18.4.59 TRNG Counter Register (CRYP0_TRNG_CNTR)

This register provides the access to the Counter register for test and diagnostic purposes. It is accessible in Test Mode only.

Reset: 0x0000 0000

Figure 18-59. TRNG Counter Register (CRYP0_TRNG_CNTR)

Bit	Mnemonic	Description	Comments
31:24		Reserved	
23:0	CNT	Counter State machine counter for test and diagnostic purposes.	

18.4.60 TRNG Alarm Counter Register (CRYP0_TRNG_ALRM)

This register accumulates the number of comparison faults detected on the output data. An output data block that is identical to the prior output generates a comparison fault. A very small number of faults are to be expected statistically. The function reading this register must apply an algorithm to determine what is significant. The count value is reset to zero whenever an output sample does not generate a comparison fault.

Reset: 0x0000 0000

Figure 18-60. TRNG Alarm Counter Register (CRYP0_TRNG_ALRM)

Bit	Mnemonic	Description	Comments
31:8		Reserved	
7:0	ALMCNT	Alarm Count Provides a non-cumulative count of the number of comparison faults that have been detected on the TRNG output.	

18.4.61 TRNG Configuration Register (CRYP0_TRNG_CFG)

This register configures the TRNG.

Reset: 0x0000 0FDC

Figure 18-61. TRNG Configuration Register (CRYP0_TRNG_CFG)

Bit	Mnemonic	Description	Comments
31:12		Reserved	
11:6	RSTCNT	Reset Count Threshold of consecutive unchanging output bits for resetting LFSRs.	The counter value is set to 0x01 to avoid having the TRNG reset immediately after start-up.
5:3	R2DLY	Ring 2 Delay Selects delay for Ring Oscillator 2	
2:0	R1DLY	Ring 1 Delay Selects delay for Ring Oscillator 1	

18.4.62 TRNG Test Read of LFSR 0 Low/High Registers (CRYP0_TRNG_LF0L/H)

These registers provide access to Linear Feedback Shift Register (LFSR) 0 for test and diagnostic purposes. They are accessible in Test Mode only.

Reset: 0x0000 0000

Figure 18-62. TRNG Test Read of LFSR 0 Low Register (CRYP0_TRNG_LF0L)

Bit	Mnemonic	Description	Comments
31:0	LFSR0	32 least significant state bits of LFSR 0 [31:0]	

Figure 18-63. TRNG Test Read of LFSR 0 High Register (CRYP0_TRNG_LF0H)

Bit	Mnemonic	Description	Comments
31:17		Reserved	
16:0	LFSR0	17 most significant state bits of LFSR 0 [48:32]	

18.4.63 TRNG Test Read of LFSR 1 Low/High Registers (CRYP0_TRNG_LF1L/H)

These registers provide access to LFSR 1 for test and diagnostic purposes. They are accessible in Test Mode only.

Reset: 0x0000 0000

Figure 18-64. TRNG Test Read of LFSR 1 Low Register (CRYP0_TRNG_LF1L)

Bit	Mnemonic	Description	Comments
31:0	LFSR1	32 least significant state bits of LFSR 1 [31:0]	

Figure 18-65. TRNG Test Read of LFSR 1 High Register (CRYP0_TRNG_LF1H)

Bit	Mnemonic	Description	Comments
31:16		Reserved	
15:0	LFSR1	16 most significant state bits of LFSR 1 [47:32]	

18.4.64 TRNG Triple DES Key 0 Low/High Registers (CRYP0_TRNG_K0_L/H)

These registers provides the Triple-DES Key 0 to the post-processor. The key register is implemented as a 64-bit maximum length LSFR to generate pseudo-random key values and is updated at the end of a TRNG operation.

Reset: 0x0000 0000

Figure 18-66. TRNG Triple DES Key 0 Low/High Registers (CRYP0_TRNG_K0_L/H)

Bit	Mnemonic	Description	Comments
31:0		Key 0 CRYP0_TRNG_K0_L – bits 31:0 (lsb) CRYP0_TRNG_K0_H – bits 63:32 (msb)	If the processor reads this register it returns an undefined value.

18.4.65 TRNG Triple DES Key 1 Low/High Registers (CRYP0_TRNG_K1_L/H)

These registers provides the Triple-DES Key 1 to the post-processor. The key register is implemented as a 64-bit maximum length LSFR to generate pseudo-random key values and is updated at the end of a TRNG operation.

Reset: 0x0000 0000

Figure 18-67. TRNG Triple DES Key 1 Low/High Registers (CRYP0_TRNG_K1_L/H)

Bit	Mnemonic	Description	Comments
31:0		Key 1 CRYP0_TRNG_K1_L – bits 31:0 (lsb) CRYP0_TRNG_K1_H – bits 63:32 (msb)	If the processor reads this register it returns an undefined value.

18.4.66 TRNG Initialization Vector Low/High Registers (CRYP0_TRNG_IV_L/H)

These registers provides an initialization vector (IV) to the post-processor. For FIPS 140-2 compliance, the DES post-processor must operate in CBC (cipher-block-chaining) mode.

Reset: 0x0000 0000

Figure 18-68. TRNG Initialization Vector Low/High Registers (CRYP0_TRNG_IV_L/H)

Bit	Mnemonic	Description	Comments
31:0		Key 1 CRYP0_TRNG_IV_L – bits 31:0 (lsb) CRYP0_TRNG_IV_H – bits 63:32 (msb)	If the processor reads this register it returns an undefined value.

18.4.67 PKA x Vector Address Register (CRYP0_PKA_x_PTR)

These registers provide the address of the x vector (x= A, B, C, or D) in the PKM.

Reset: 0x0000 0000

Figure 18-69. PKA x Vector Address Register (CRYP0_PKA_x_PTR)

Bit	Mnemonic	Description	Comments
31:11		Reserved	
10:0	xADDR (x = A, B, C, or D)	x Address Address of the least significant word of the x vector.	

18.4.68 PKA x Vector Length Register (CRYP0_PKA_x_LEN)

These registers provide the length of the x vector (x= A or B) in the PKM.

Reset: 0x0000 0000

Figure 18-70. PKA x Vector Length Register (CRYP0_PKA_x_LEN)

Bit	Mnemonic	Description	Comments
31:9		Reserved	
8:0	xL (x = A or B)	x Length Length of the x vector in 32-bit words.	Valid settings range from 0 – 512.

18.4.69 PKA Shift Register (CRYP0_PKA_SHIFT)

This register provides the operation code for the public key operation.

Reset: 0x0000 0000

Figure 18-71. PKA Shift Register (CRYP0_PKA_SHIFT)

Bit	Mnemonic	Description	Comments
31:5		Reserved	
4:0	SV	Shift Value Specifies the number of bits to shift in the arithmetic operation	

18.4.70 PKA Function Code Register (CRYP0_PKA_FUNC)

This register provides the function code for the public key operation.

Note: Continuously reading this register to poll the RUN bit is *not* allowed with executing complex Sequencer operations (the Sequencer cannot access the PKCP when this is done). Ensure there is at least one SysClk cycle between poll operations.

Reset: 0x0000 0000

Figure 18-72. PKA Function Code Register (CRYP0_PKA_FUNC)

Bit	Mnemonic	Description	Comments
31:25		Reserved	
24	STLR	Stall Result	Pauses update of CRYP0_PKA_COMP, CRYP0_PKA_MSW and CRYP0_PKA_DIV registers as well as resetting the Run bit. Allows software to read results from a previous operation when the newly started operation is completed quickly. Used only for *basic* PKA operations.
23:16		Reserved	
15	RUN	Run 0 Operation is complete 1 Start the requested operation (bits 14:0)	
14:12	SQOP	Sequencer Operations 000 None 001 ExpMod-CRT 010 ExpMod-ACT4 (compatible with EIP23) 011 ECC-ADD (if available in firmware; otherwise, reserved) 100 ExpMod-ACT2 (compatible with EIP23) 101 ECC-Mul (if available in firmware; otherwise, reserved) 110 ExpMod-variable 111 ModInv (if available in firmware; otherwise, reserved)	The encoding of these operations is determined by Sequencer firmware.
11	CPY	Copy	
10	CPR	Compare	
9	MOD	Modulo	
8	DIV	Divide	
7	LSHFT	Left Shift	
6	RSHFT	Right Shift	
5	SUB	Subtract	
4	ADD	Add	
3	MSO	Most Significant One	Loads location of MS one in result word.
2		Reserved	
1	DMLT	Add/Subtract	Perform combined Add/Subtract operation.
0	MLT	Multiply	

The following tables describe the operations associated with processing security vectors using the settings in the above register.

Table 18-18. Mathematical Operations

Function	Operation	Vector A	Vector B	Vector C	Vector D
Multiply	A x B -->C	Multiplicand	Multiplier	Product	na
Add	A + B --> C	Addend	Addend	Sum	na
Subtract	A – B --> C	Minuend	Subtrahend	Difference	na
AddSub	A + C – B --> D	Addend	Subtrahend	Addend	Result
Right Shift	A >> Shift --> C	Input	na	Result	na
Left Shift	A << Shift --> C	Input	na	Result	na
Divide	A mod B -->C A div B --> D	Dividend	Divisor	Remainder	Quotient
Modulo	A mod B --> C	Dividend	Divisor	Remainder	na
Compare	A = B, A < B, A > B	Input1	Input2	na	na
Copy	A --> C	Input	na	Result	na

Table 18-19. Operational Restrictions

Function	Requirements
Multiply	0 < A_Len, B_Len ≤ Max_Len.
Add	0 < A_Len, B_Len ≤ Max_Len.
Subtract	0 < A_Len, B_Len ≤ Max_Len (result must be positive; i.e., A ≥ B).
AddSub	0 < A_Len, B_Len ≤ Max_Len (B and C have A_Len; B_Len is ignored).
Right Shift	0 < A_Len ≤ Max_Len.
Left Shift	0 < A_Len ≤ Max_Len.
Divide, Modulo	1< B_Len ≤ A_Len ≤ Max_Len (most significant word of B operand cannot be 0).
Compare	0 < A_Len ≤ Max_Len (B operand has A_Len; B_Len is ignored).
Copy	0 < A_Len ≤ Max_Len.

Table 18-20. Result Vector Memory Allocation

Function	Result Vector	Result Vector Length (32-bit Words)
Multiply	C	A_Len + B_Len + 6 (6 words are scratch pad and should be discarded).
Add	C	Max(A_Len, B_Len) + 1.
Subtract	C	Max(A_Len, B_Len).
AddSub	D	A_Len + 1.
Right Shift	C	A_Len.
Left Shift	C	A_Len + 1 (shift is non-zero).
Divide	C D	Remainder is B_Len + 1 (1 word is scratch pad and should be discarded) Quotient is A_Len – B_Len + 1.
Modulo	C	Remainder is B_Len + 1.
Compare	none	Updates the PKA_Compare register.
Copy	C	A_Len.

Table 18-21. Result Vector/Input Vector Overlap Restrictions

Function	Result Vector	Restrictions
Multiply	C	No overlap with A or B vectors allowed.
Add, Subtract	C	Can overlap with A and/or B, provided the start address of the C vector is not above the start address of the vectors.
AddSub	D	Can overlap with A, B, and/or C, provided the start address of the D vector does not lie above the start address of the vectors which it overlaps.
Right/Left Shift	C	Can overlap with A vector, provided the start address of the C vector is not above the start address of the A vector.
Divide	C D	Cannot overlap A, B, or D vectors. Cannot overlap A, B, or C vectors.
Modulo	C	Cannot overlap A or B vectors.
Compare	none	
Copy	C	Can overlap with A vector, provided the start address of the C vector is not above the start address of the A vector. Copy of a vector to a lower address is always allowed even if source and destination overlap.

Table 18-22. Modular Exponentiation Operations

Function	Operation	Vector A	Vector B	Vector C	Vector D
ExpMod–ACT2 ExpMod–ACT4 ExpMod–variable	C mod B --> D	Exponent len = A_Len	Modulus len = B_Len	Base len = B_Len	Result & Workspace
ExpMod–CRT	X <-- (Input mod Mod P)^P mod Mod P Y <-- (Input mod Mod Q)^Q mod Mod Q Z <-- ((((X - Y)mod Mod P) * Q inverse) mod Mod P) * Mod Q Result <-- Y + Z	Exp P followed by Exp Q at next higher even address. Both len = A_Len	Mod P + buffer followed by Mod Q at next higher word address. Both len = B_Len	Q inverse len = B_Len	Input, Result. Both are 2xB_Len & Workspace

18.4.71 PKA Comparison Result Register (CRYP0_PKA_COMP)

This register provides the result of Compare operations.

Reset: 0x0000 0000

Figure 18-73. PKA Comparison Result Register (CRYP0_PKA_COMP)

Bit	Mnemonic	Description	Comments
31:3		Reserved	
2	SUP	A > B 0 A ≥ B 1 A > B	
1	INF	A < B 0 A ≤ B 1 A < B	
0	EQL	A = B 0 A ≠ B 1 A = B	

18.4.72 PKA Quotient MSW Register (CRYP0_PKA_DIV)

This register returns the PKM address of the most significant, non-zero 32 bit word of the quotient vector. For an all-zero result vector, bit 11 is asserted, and the MSW address should be ignored. This register is only applicable to divide operations.

The divide operation presents a special case because two result vectors are generated—the quotient and remainder (modulus). This register corresponds to the quotient vector. The MSW of the remainder vector is specified in the CRYP0_PKA_MOD register.

Reset: 0x0000 8000

Figure 18-74. PKA Quotient MSW Register (CRYP0_PKA_DIV)

Bit	Mnemonic	Description	Comments
31:16		Reserved	
15	RZERO	Result Zero 0 Non-zero quotient vector 1 Zero quotient vector	
14:11		Reserved	
10:0	DIVMSW	DIV MSW Address PKM address of the most significant, non-zero 32-bit word of the quotient vector.	

18.4.73 PKA Remainder MSW Register (CRYP0_PKA_MOD)

This register returns the PKM address of the most significant, non-zero 32-bit word of the remainder vector. For an all-zero remainder vector, bit 11 is asserted, and the MSW address should be ignored. The results of this register is valid for basic Divide and Modulo operations.

The divide operation presents a special case because two result vectors are generated—the quotient and remainder (modulus). This register corresponds to the remainder vector. The MSW of the quotient vector is specified in the CRYP0_PKA_DIV register.

Reset: 0x0000 0800

Figure 18-75. PKA Remainder MSW Register (CRYP0_PKA_MOD)

Bit	Mnemonic	Description	Comments
31:16		Reserved	
15	RZERO	Result Zero 0 Non-zero result vector 1 Zero result vector	
14:11		Reserved	
10:0	RMSWA	Result MSW Address PKM address of the most significant, non-zero 32-bit word of the result vector.	

18.4.74 PKA Remainder MSW Register (CRYP0_PKA_MOD)

This register returns the PKM address of the most significant, non-zero 32-bit word of the remainder vector. For an all-zero remainder vector, bit 11 is asserted, and the MSW address should be ignored. The results of this register is valid for basic Divide and Modulo operations.

The divide operation presents a special case because two result vectors are generated—the quotient and remainder (modulus). This register corresponds to the remainder vector. The MSW of the quotient vector is specified in the CRYP0_PKA_DIV register.

Reset: 0x0000 0800

Figure 18-76. PKA Remainder MSW Register (CRYP0_PKA_MOD)

Bit	Mnemonic	Description	Comments
31:16		Reserved	
15	RZERO	Result Zero 0 Non-zero result vector 1 Zero result vector	
14:11		Reserved	
10:0	RMSWA	Result MSW Address PKM address of the most significant, non-zero 32-bit word of the result vector.	

18.4.75 PKA Sequencer Control/Status Register (CRYP0_PKA _SEQ)

The sequencer is accessed through this single control/status register. With the exception of bit 31, the use of this register is determined by the Sequencer firmware. This register should only be accessed when the Sequencer program is stored in RAM.

Access: See register field.

Reset: 0x0000 0000

Figure 18-77. PKA Sequencer Control/Status Register (CRYP0_PKA _SEQ)

Bit	Mnemonic	Description	Comments
31	RESET	Reset 0 Start sequencer program 1 Reset state	R/W
30:16		Reserved	
15:8	SEQST	Used by the sequencer to communicate program status.	Read only
7:0	CTLTRG	Sequencer state Set to 1 triggers/controls state of sequencer.	R/W Only Sequencer can reset to 0.

Example:

The following is an example of how to use the PKA multiply function:

The security function is little-endian, so that is how the data must be applied and the result interpreted.

The A operand data = 0x0000 0003 0000 0001 (where 01 is LSB)
The B operand data = 0x0000 0004 0000 0002 (where 02 is LSB)

PKA_A_PTR = 0x390
PKA_B_PTR = 0x27E
PKA_C_PTR = 0x0BA

PKA_A_LENGTH = 0x2
PKA_B_LENGTH = 0x2

PKA Memory

0x390* = 0x00000001
0x391 = 0x00000003

0x27E = 0x00000002
0x27F = 0x00000004

0x0BA = 0x00000002
0x0BB = 0x0000000A
0x0BC = 0x0000000C
0x0BD = 0x00000000

(*word addresses)
So the final C operand data = 0x0000 0000 0000 000C 0000 000A 0000 0002
(where 02 is LSB)

Compare with the paper calculation method:

```
                    0x0000000300000001
                    0x0000000400000002
                    ---------------------------
                    0x0000000600000002
          + 0x0000000C0000000400000000
-----------------------------------------------------
0x000000000000000C0000000A00000002
```

18.4.76 Interrupt Unmasked and Masked Status Registers (CRYP0_INT_UNMSK/MSK)

These registers provide interrupt status visibility to the processor. CRYP0_INT_UNMSK applies *prior* to the interrupt mask being applied. CRYP0_INT_MSK applies *after* to the interrupt mask is applied. Using CRYP0_INT_UNMSK, the processor can view all potential sources of incoming interrupts. Using CRYP0_INT_MSK, the processor can view all interrupts directed to the UIC. All of these sources, whether masked in or out, will be latched in these registers and must be cleared using the CRYP0_INT_MSK register in order to capture a subsequent event. A bit set to 1 indicates that the associated interrupt is present.

Access: Read only – CRYP0_INT_UNMSK
R/W – CRYP0_INT_MSK (Reading views the interrupts. Writing clears the interrupts. Writing 1 to a bit clears the corresponding interrupt. Writing 0 leaves the interrupt latch unchanged.)

Reset: 0x0000 0000

Figure 18-78. Interrupt Unmasked and Masked Status Registers (CRYP0_INT_UNMSK/MSK)

Bit	Mnemonic	Description	Comments
31:19		Reserved	
18	TMOUTERR	Exceeded timeout counter limit	
17	MAWRERR	Error during Bus Master write	
16	MARDERR	Error during Bus Master read	
15	DESC	Command to the Packet Engine to fetch the next Packet Descriptor from the PDR.	Applicable only when the Packet Fetch Engine has been enabled with a non-0 CRYP0_PE_RING_S register setting. It is not an internally generated interrupt. It is used for test purposes only to verify that the DESC command in SDR0_CRYP0 has been received correctly by the packet engine.
14	DESCRD	Specified number of Descriptors (1–63) have completed processing. Also occurs if a Descriptor has just been processed and there are no more input Descriptors available.	Not available when Descriptor Ring processing is disabled.
13	CNTXT	Processing of a Descriptor written to the Command Queue has finished	Asserted directly after the result Descriptor is written. Asserted only when the Packet Descriptor Ring is disabled.
12	CMD	Processing of a Descriptor written to the Command Queue has finished	Asserted directly after the result Descriptor is written. Asserted only when the Packet Descriptor Ring is disabled.
11	DATAO	Packet Engine is ready to *return* data fragment of size	
10	DATAI	Packet Engine is ready to *receive* data fragment of size	
9	PEERR	Packet Engine processing error	
8	UDMS	User-initiated Master DMA transaction complete	
7	DMAAV	User DMA command registers are available	DMA transaction has been loaded and the holding registers are available.
6	DMAERR	Error occurred during a user-initiated DMA transfer	

Figure 18-78. Interrupt Unmasked and Masked Status Registers (CRYP0_INT_UNMSK/MSK)			
Bit	**Mnemonic**	**Description**	**Comments**
5	PKT	Packet Engine finished processing of a complete packet	All remaining data is read from the output buffer.
4	SLVERR	Illegal transfer to the PLB Slave requested	
3	MSTERR	Timeout occurred on a PLB Master transfer request	
2	TRNG	TRNG finished generation of a True Random Number	
1	PRNG	PRNG finished generation of a Pseudo Random Number	
0	PKA	PKA finished an operation	

18.4.77 Interrupt Mask Register (CRYP0_INT_EN)

This register configures the interrupt mask for the UIC. Writing 1 will enable the interrupt source and 0 will disable it. When an interrupt source is disabled the matching interrupt is cleared in the CRYP0_INT_MSK register.

Reset: 0x0000 0000

Figure 18-79. Interrupt Mask Register (CRYP0_INT_EN)			
Bit	**Mnemonic**	**Description**	**Comments**
31:19		Reserved	
18	TMOUTERR	Exceeded timeout counter limit	
17	MAWRERR	Error during Bus Master write	
16	MARDERR	Error during Bus Master read	
15	DESC	Specified number of Descriptors (1–63) have completed processing. Also occurs if a Descriptor has just been processed and there are no more input Descriptors available.	Not available when Descriptor Ring processing is disabled.
14	DESCRD	Command to the Packet Engine to fetch the next Packet Descriptor from the PDR.	Applicable only when the Packet Fetch Engine has been enabled with a non-0 CRYP0_PE_RING_S register setting.
13	CNTXT	Processing of a Descriptor written to the Command Queue has finished	Asserted directly after the result Descriptor is written. Asserted only when the Packet Descriptor Ring is disabled.
12	CMD	Processing of a Descriptor written to the Command Queue has finished	Asserted directly after the result Descriptor is written. Asserted only when the Packet Descriptor Ring is disabled.
11	DATAO	Packet Engine is ready to *return* data fragment of size	
10	DATAI	Packet Engine is ready to *receive* data fragment of size	
9	PEERR	Packet Engine processing error	
8	UDMS	User-initiated Master DMA transaction complete	

Figure 18-79. Interrupt Mask Register (CRYP0_INT_EN) (Continued)			
Bit	**Mnemonic**	**Description**	**Comments**
7	DMAAV	User DMA command registers are available	DMA transaction has been loaded and the holding registers are available.
6	DMAERR	Error occurred during a user-initiated DMA transfer	
5	PKT	Packet Engine finished processing of a complete packet	All remaining data is read from the output buffer.
4	SLVERR	Illegal transfer to the PLB Slave requested	
3	MSTERR	Timeout occurred on a PLB Master transfer request	
2	TRNG	TRNG finished generation of a True Random Number	
1	PRNG	PRNG finished generation of a Pseudo Random Number	
0	PKA	PKA finished an operation	

18.4.78 Interrupt Configuration Register (CRYP0_INT_CFG)

This register configures the interrupt type which will be provided to the UIC.

Configuring the Interrupt output type for *Pulse* causes the interrupt signal to pulse low for two clock cycles when activated. When set for *Level*, the interrupt signal is set low until cleared by the processor (that is, it will follow the bit in the Masked Status Register). For the PLB bus, this is typically set to Level.

Reset: 0x0000 0000

Figure 18-80. Interrupt Configuration Register (CRYP0_INT_CFG)			
Bit	**Mnemonic**	**Description**	**Comments**
31:2		Reserved	
1	IPCLR	Pulse interrupt clear 0 Interrupt must be reset by processor 1 Reset pulse interrupt	Used only when bit 0 is 1.
0	IHOT	Interrupt output type 0 Level 1 Pulse	

18.4.79 Interrupt Force Descriptor Read Register (CRYP0_INT_DESRD)

This register allows the processor to force the packet engine to read the next descriptor or block of descriptors. The data contents of the write operation are ignored. Any write to this register will cause the descriptor fetch. This register is only applicable when the External Descriptor Ring Handler is enabled (PE ring size is non-0).

If the Host reads this register, it returns zeros.

Reset: 0x0000 0000

Figure 18-81. Interrupt Force Descriptor Read Register (CRYP0_INT_DESRD)

Bit	Mnemonic	Description	Comments
31:0		Don't care	

18.4.80 Interrupt Descriptor Count Register (CRYP0_INT_DESCT)

This register configures the number of Descriptors that must be completed before issuing a PE Descriptor Done interrupt. Valid settings range from 1–63.

Note: If the Packet Engine has processed at least one Descriptor and then has no valid Descriptor in its internal Command Queue, it issues a P/E Descriptor Done interrupt immediately.

Reset: 0x0000 0001

Figure 18-82. Interrupt Descriptor Count Register (CRYP0_INT_DESCT)

Bit	Mnemonic	Description	Comments
31:6		Reserved	
5:0	DCCNT	Descriptor Complete Count	A setting of 0 is invalid and is interpreted as 1.

Example:

Assume the CRYP0_INT_DESCT register is set to 0x0004. If three packets arrive and are queued on the PDR, the security processor processes them and populates three result descriptors on the RDR. However, an interrupt will *not* be generated, since the counter is waiting until a fourth packet is processed. If some time passes before a fourth packet arrives, the three completed packets stall in the result queue without the processor being aware of them. The processor can predict this scenario and poll for Result Descriptors without waiting for an interrupt.

18.4.81 Interrupt Timeout Count Register (CRYP0_INT_TMO)

This register allows configuring a number of clock cycles before issuing a timeout interrupt. The counter can be used together with the descriptor done counter to control communication with the packet engine and monitor packet completion.

Reset: 0x0000 0000

Figure 18-83. Interrupt Timeout Count Register (CRYP0_INT_TMO)

Bit	Mnemonic	Description	Comments
31:10	TMOCNT	Number of 1024 cycle increments	Value that defines a fixed number of clock cycles before a timeout interrupt is generated. Valid settings range from 1 to 8388607 in increments of 1024 clock cycles. The count indicates how many increments of 1024 clock cycles are allowed to occur before a timeout is generated. Zero will disable the timeout counter.
9:0		Reserved	

18.4.82 Device Control Register (CRYP0_DC_CTRL)

This register contains the device control information.

Reset: 0x0001 0001

Figure 18-84. Device Control Register (CRYP0_DC_CTRL)

Bit	Mnemonic	Description	Comments
31:18		Reserved	
17	TRNGE	TRNG Enable 0 TRNG is held in reset 1 Enable TRNG	It is useful to hold the TRNG in reset to conserve power if this function is not being used.
16	PKAE	PKA Enable 0 PKA held in reset 1 Enable PKA	It is useful to hold the PKA in reset to conserve power if this function is not being used.
15:1		Reserved	
0	TDESE	Triple-DES Enable 0 Attempts to execute a Triple-DES operation in the PE result in a Prohibited Algorithm status error 1 Enable Triple-DES encryption algorithm	

18.4.83 Device ID Register (CRYP0_DC_DEVID)

This register contains the device and vendor ID information.

Reset: 0x16AE 0094

Figure 18-85. Device ID Register (CRYP0_DC_DEVID)

Bit	Mnemonic	Description	Comments
31:16	VID	Vendor ID 0x16AE	
15:0	DID	Device ID 0x0094.	

18.4.84 Device Information Register (CRYP0_DC_DEVINF)

This register provides an indication of the security function features available in this instantiation. If a feature is enabled in this security function version, the corresponding bit is set to 1. If the feature is disabled, the corresponding bit is 0.

Reset: 0x0308 7722

Figure 18-86. Device Information Register (CRYP0_DC_DEVINF)

Bit	Mnemonic	Description	Comments
31:26		Reserved	
25	KF9	KASUMI f9	
24	KKF8	KASUMI/KASUMI f8	
23:20		Reserved	
19	PRNG	Pseudo Random Number Generator	
18:15		Reserved	
14	SHA2	SHA-2 Hash	
13	SHA1	SHA-1 Hash	
12	MD5	MD5 Hash	
11		Reserved	
10	AES	AES (Rijndael) Encryption	
9	ARC4	ARC4 Encryption	
8	DES	DES/Triple-DES encryption	
7:4	MAJ	Revision Upper and lower nibbles define the Major and Minor Revisions of the security function.	The default revision is 2.2.
3:0	MIN		

18.4.85 DMA Source Address Register (CRYP0_DMA_USRC)

This register contains the DMA source address.

Reset: 0x0000 0000

Figure 18-87. DMA Source Address Register (CRYP0_DMA_USRC)

Bit	Mnemonic	Description	Comments
31:0		DMA Source Address	

18.4.86 DMA Destination Address Register (CRYP0_DMA_UDST)

This register contains the DMA destination address.

Reset: 0x0000 0000

Figure 18-88. DMA Destination Address Register (CRYP0_DMA_UDST)

Bit	Mnemonic	Description	Comments
31:0		DMA Destination Address	

18.4.87 DMA Command Register (CRYP0_DMA_UCMD)

This register specifies the transfer length and the Bus IDs for the data source and destination. After writing this register, the transaction will be queued into the 4-channel DMA controller and is executed when the DMA scheduler passes control to the DMA channel.

Reset: 0x8000 0000

Figure 18-89. DMA Command Register (CRYP0_DMA_UCMD)

Bit	Mnemonic	Description	Comments
31:16		Reserved	
15:14	SRC	Transfer direction 0110 From external memory to the security function 1001 From the security function to external memory All other bit combinations are reserved.	
13:12	DST		
11:0	TXL	Transfer Length Data length in bytes for the DMA transaction	Valid values range from 1 – 4095.

18.4.88 DMA Configuration/Status Register (CRYP0_DMA_CFG

The register is used to specify the maximum burst transfer size settings for master and slave and enabling incremental and locked transfers on the PLB.

Reset: 0x0018 0006

Figure 18-90. DMA Configuration/Status Register (CRYP0_DMA_CFG)

Bit	Mnemonic	Description	Comments
31:10		Reserved	
9:8	PLBP	PLB priority 00 Very low 01 Low 10 High 11 Very high	Priority of PLB requests by the PLB master interface to the PLB arbiter
7:5		Reserved	
4	SREQ	Secondary request 0 Master does no secondary requests 1 NA This instantiation is fixed at SREQ = 0.	Read only
3:2		Reserved	
1:0	MBS	Maximum burst size in bytes at the PLB 00 32 (2 beats on the PLB) 01 64 (4 beats on the PLB) 10 128 (8 beats on the PLB) 11 256 (16 beats on the PLB) This instantiation is fixed at MBS = 11.	Read only When there is less data available then the maximum burst size, the length of the burst is less than the maximum burst size. Any requested transfers larger than this size will be broken up into multiple PLB Master burst transfers of this size or less.

18.4.89 PRNG Status Register (CRYP0_PRNG_STAT)

The register contains the status of the PRNG. Result Ready is mode dependent and is set only in Manual mode (CRYP0_PRNG_CTRL[AUTO] = 0). Busy is mode independent.

Reset: 0x0000 0000

Figure 18-91. PRNG Status Register (CRYP0_PRNG_STAT)

Bit	Mnemonic	Description	Comments
31:2		Reserved	
1	RRDY	Result Ready 0 PRNG enabled for the next operation 1 Valid PRN available in the result register.	Set only when the PRNG is controlled by software, and CRYP0_PRNG_CTRL[AUTO] = 0.
0	BSY	Busy PRNG is busy 0 Valid PRN available in the output register. 1 CRYP0_PRNG_CTRL[E] = 1	This bit is also set when in Auto mode (CRYP0_PRNG_CTRL[AUTO] = 1) and the PRNG is busy In Auto mode this bit is cleared after a number has been generated, and the PRNG is waiting for the PE to use the PRN.

18.4.90 PRNG Control Register (CRYP0_PRNG_CTRL)

The register contains control settings for generation of PRNs.

Reset: 0x0000 0000

Figure 18-92. PRNG Control Register (CRYP0_PRNG_CTRL)

Bit	Mnemonic	Description	Comments
31:3		Reserved	
2	R128	Result 128 Enables the generation of 128-bit Pseudo-Random Numbers. 0 PRNG fills only the lower 64 bits of the result register (CRYP0_PRNG_RES0). 1 PRNG fills both 64-bit result registers, starting with the lower 64 bits (CRYP0_PRNG_RES0) and ending with the upper 64 bits (CRYP0_PRNG_RES1). This option reduces the performance by half.	A read always returns the last written value.
1	AUTO	Auto Enables the automatic generation of PRNs. 0 PRNG stops the automatic generation of a PRNs at completion of the PRNG algorithm. Can be cleared only when CRYP0_PRNG_STAT[BSY] = 0. 1 PRNG starts the automatic generation of 128-bit PRNs when CRYP0_PRNG_CTRL[E] = 1. This is a continuous operation until CRYP0_PRNG_CTRL[AUTO] = 0.	This mode can be used for generating IV's for the DES and AES algorithms. The length of the IV is automatically adjusted for the algorithm. The PRNG is under control of the PE and generates a new IV when required. A read always returns the last written value.
0	E	Enable Enables the generation of a new PRN 0 PRNG has started an operation and CRYP0_PRNG_CTRL[AUTO] = 0. 1 PRNG starts the generation of a PRN.	A write with zero has no effect. Always read as zero.

18.4.91 PRNG Seed Value L/H Registers (CRYP0_PRNG_SDL/H)

These registers contains the 64-bit secret seed value that is seeded once after reset and then automatically updated with the output of the third Triple-DES operation.

Reset: 0x0000 0000

Figure 18-93. PRNG Seed Value L/H Registers (CRYP0_PRNG_SDL/H)

Bit	Mnemonic	Description	Comments
31:0		Seed Value 0:31 – CRYP0_PRNG_SDL 63:32 – CRYP0_PRNG_SDH	

18.4.92 PRNG Key x Low/High Registers (CRYP0_PRNG_KxL/H)

There are two key register pairs numbered 0 and 1 (x). These four registers contain the 64-bit cryptograhic keys used for Triple-DES operations. Bit 31:0 are in the low (L) register while bits 63:32 are in the high (H) register.

The DES keys are 64-bits long, and include eight parity-check bits (bits on positions 0, 8, 16, 24, 32, 40, 48 and 56). The KEY registers are implemented using a 56-bit maximum-length LFSR implementation, ignoring the parity bits. The KEY registers are updated (LFSR changes to next value) after every PRNG operation. These registers should *not* contain all zeros.

Reset: 0x0000 0000

Figure 18-94. PRNG Key x Low/High Registers (CRYP0_PRNG_KEYx_L/H)

Bit	Mnemonic	Description	Comments
31:0		PRNG Key Key bits 63:32 are in CRYP0_PRNG_KxH. Key bits 31:0 are in CRYP0_PRNG_KxL.	If the processor reads these registers, they return an undefined value.

18.4.93 PRNG Result x Registers (CRYP0_PRNG_RSx)

These registers contain two generated PRNs. A pair of result registers are concatenated to form one unique 128-bit result. The contents of the result registers are valid when CRYP0_PRNG_STAT[BSY] = 0 or the PRGN interrupt signal is high.

Reset: 0x0000 0000

Figure 18-95. PRNG Result x Registers (CRYP0_PRNG_RSx)

Bit	Mnemonic	Description	Comments
31:0		PRNG Result Bits 31:0 are in CRYP0_PRGN_RS0 Bits 63:32 are in CRYP0_PRGN_RS1 Bits 95:64 are in CRYP0_PRGN_RS2 Bits 127:96 are in CRYP0_PRGN_RS3	

18.4.94 PRNG LFSR Low/High Registers (CRYP0_PRNG_LFL/H)

These registers contain a unique input DT as plain text input for the first Triple-DES operation. The register is implemented using a 64-bit maximum length LFSR. The register is updated (LFSR changes to next value) after every PRNG operation. These registers should *not* contain all zeros.

Reset: 0x0000 0000

Figure 18-96. PRNG LFSR Low/High Registers (CRYP0_PRNG_LFL/H)

Bit	Mnemonic	Description	Comments
31:0		CRYP0_PRNG_LFH – 32 most significant state bits [63:32] of the LFSR CRYP0_PRNG_LFL – 32 least significant state bits [31:0] of the LFSR	Reading these registers return the current value of the LFSR.

18.5 Buffer Interface

The security function has three buffers for use in processing data streams. Each of the buffers is described in detail in the following sections.

18.5.1 ARC4 Buffer

The ARC4 RAM buffer is used to store the pre-processed key that initializes the ARC4 function. Consists of a 256 Bytes write-only buffer. The packet input data is written here to be transferred to the Packet Engine.

The buffer is located at 0x4 0019 0700 – 0x4 0019 07FF. In Direct Host DMA mode, the processor must start writing from the base address (0x4 0019 0700) and increment the address pointer for each write.

A read from any address in the address range of the ARC4 Buffer returns an undefined value.

18.5.2 Input buffer

This is the location where input data is written into the security function in the Host DMA mode. The processor monitors the available space in the input buffer by means of the CRYP0_PE_DMA_ST register (see page 356).

This is a 2KB write-only buffer. The packet input data is written here to be transferred to the Packet Engine. The input buffer is located at 0x4 0019 8000 – 0x4 0019 87FF.

In Direct Host DMA mode, the processor must start writing from the base address and increment the address pointer for each write. If a packet exceeds 2KB, the buffer will not be large enough and part of the buffer must be reused (wrapped back to the base address). The processor must monitor the CryptCore Input Size field (CRYP0_PE_DMA_ST[SEIS]) to keep track of the empty bytes available. To make data transfer from the processor memory to the Packet Engine more efficient, the processor can make use of Input Request Active, bit 10, in the CRYP0_PE_DMA_ST register. This bit indicates that the input buffer has N words free for write, where N is specified by the Packet Engine in the Input Threshold field (bits 9:0) of the CRYP0_PE_IO_THR register.

Example:

Assume a packet of 2400 bytes must be written to the input buffer, and the Input Threshold is set to 128 bytes. The processor starts writing to the input buffer at the base address 0x4 0019 8000 and keeps incrementing the address by 4 for every word write until 2048 bytes are written. Then the processor reads the CryptCore Input Size field in the CRYP0_PE_DMA_ST register to find out how many bytes are available in the input buffer. When space is available, the address is wrapped and the next 400 bytes are written from address 0x4 0019 8000 and onwards.

A read from any address in the address range of the input buffer returns an undefined value.

18.5.3 Output Buffer

This is the location (0x4 0019 C000 – 0x4 0019 C7FF) where output data is read from the security function in the Host DMA mode. The processor monitors the available bytes in the output buffer by means of the CRYP0_PE_DMA_ST register (see page 356).

It consists of a 2KB read-only buffer. The packet output data is read from here to be transferred to the processor memory. The processor must start reading from the base address (0x4 0019 C000) and increment the address pointer for each read. If a packet exceeds 2KB, the buffer will not be large enough and part of the buffer must be reused (wrapped back to the base address). The processor must monitor the CryptCore Output Size field in the CRYP0_PE_DMA_ST register to keep track of the filled bytes available in the Output Buffer. To make data transfer from the Packet Engine to the Host Memory more efficient, the processor can make use of the Output Request Active, bit 11 in the CRYP0_PE_DMA_ST register. This bit indicates that the Output Buffer contains N words available for read, where N is specified by the Output Threshold field (bits 25:16) in the CRYP0_PE_IO_THR register.

A write to any address in the address range of the Output Buffer has no effect.

18.5.4 Public Key Memory (PKM)

RAM for Public Key input and output vectors. This memory is located at 0x4 001C 1000 – 0x4 001C 1FFF

19. PCI Controller (PPC460EX-N and PPC460GT-N Only)

The PLB/PCI bridge implemented in PPC460EX/GT provides an interface between the PCI bus and the Processor Local Bus (PLB). It enables PCI initiators to access PLB slaves and PLB masters to access PCI targets. PCI initiators and PCI targets are in PCI conventional mode.

The PLB/PCI bridge contains an internal register set that can be accessed by both PLB masters and PCI initiators. It has many optional and programmable features that enable it to be configured for different applications. These options and features are set by using the internal registers.

The PLB/PCI bridge may be used in "host-bridge" mode or "adapter-bridge" mode. It is a "bridge" device, in that it only generates transfers as a master on the PLB or the PCI bus in response to decoding a transfer as a slave on the other bus.

19.1 PCI Features

Features include:

- Frequency to 66MHz
- 32-bit bus
- PCI Host Bus Bridge or an Adapter Device's PCI interface
- Internal PCI arbitration function, supporting up to four external devices, that can be disabled for use with an external arbiter
- Support for inbound and outbound Message Signaled Interrupts (MSI)
- Simple message passing capability
- Asynchronous to the PLB
- PCI Power Management 1.1
- PCI register set addressable both from on-chip processor and PCI device sides
- Ability to boot from PCI bus memory
- Error tracking/status
- Supports initiation of transfer to the following address spaces:
  - Single beat I/O reads and writes
  - Single beat and burst memory reads and writes
  - Single beat configuration reads and writes (type 0 and type 1)
  - Single beat special cycles
- Vital Product Data (VPD) support

Figure 19-1 describes the internal structure of the PLB/PCI bridge while *Figure 19-2* describes a typical application using PLB/PCI bridge. The PLB/PCI bridge enables PCI initiators to access PLB slaves and PLB masters to access PCI targets. PCI initiators and PCI targets are in PCI conventional mode.

Figure 19-1. PLB/PCI Bridge Internal Structure

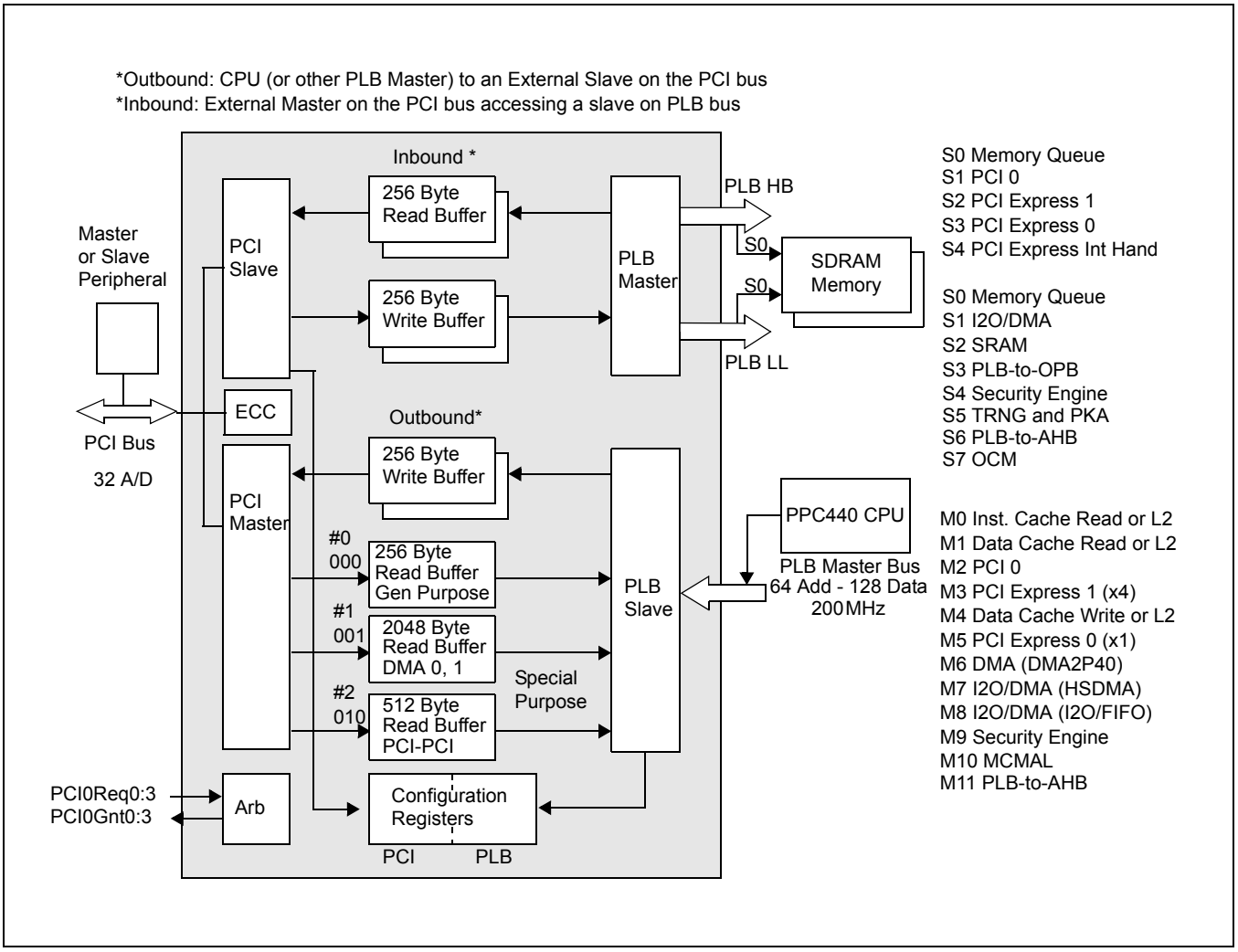
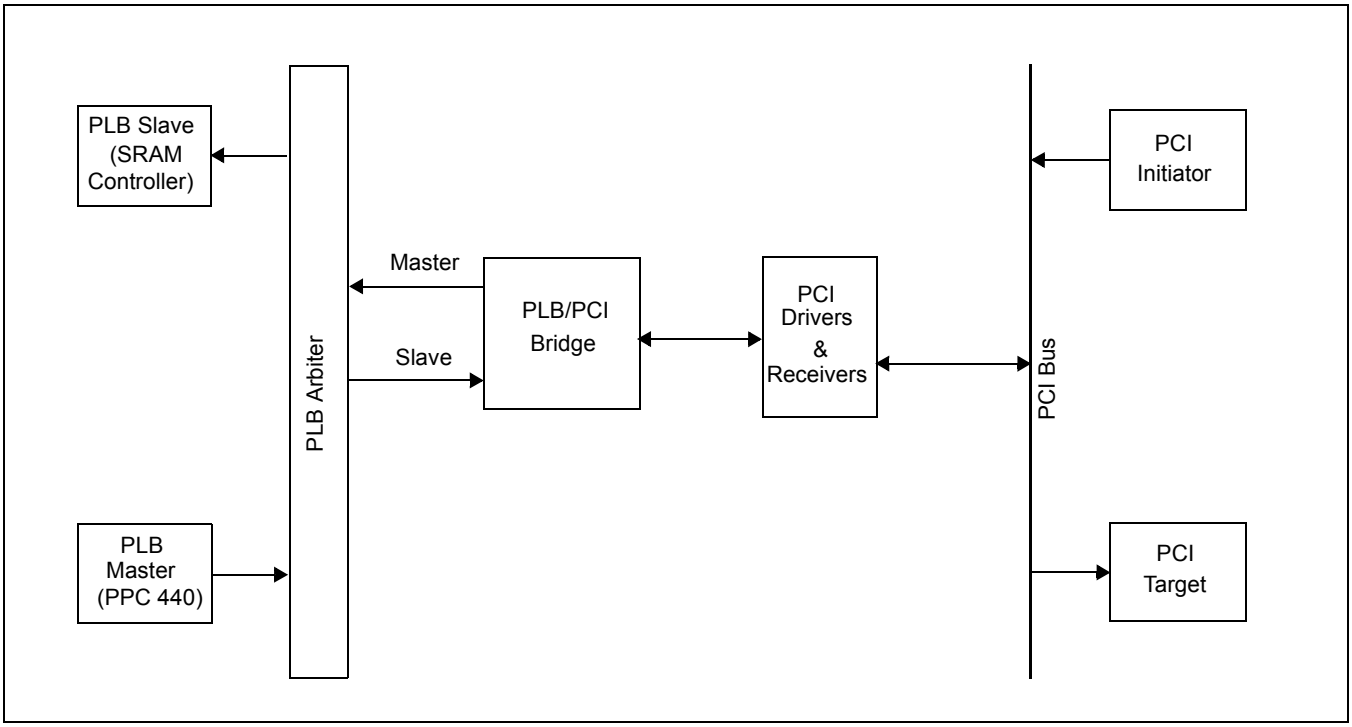


Figure 19-2. Typical Application using the PLB/PCI Bridge



The PCI Bus Controller has different Device IDs depending on the option selected.

Table 19-1. Options

Option	Device ID
PPC460GT	0xA000
PPC460GT w/o Security	0xA001
PPC460EX	0xA002
PPC460EX w/o Security	0xA003

19.2 References

- *Processor Local Bus Specification, Version 4*
- *PCI Local Bus Specification,* Version 2.3
- *PCI Bus Power Management Interface Specification*, *Version 1.1*
- *Intelligent I/O Architecture Specification*, *Version 1.5*

19.3 Inbound Transaction Handling

The PLB/PCI Bridge can claim a variety of inbound transaction types on the PCI bus and subsequently master a transfer on the PLB bus or access an internal register.

Note: This chapter defines the terms *outbound* and *inbound* to describe the direction of traffic through the PLB-PCI bridge. The PLB bus is considered *in* and the PCI bus is considered *out*. Thus, an outbound transfer is a read or write in which the PLB-PCI Bridge is the slave on the PLB bus and the initiator on the PCI bus. An inbound transfer is a read or write in which the PLB-PCI Bridge is the target on the PCI bus and the master on the PLB bus.

The PLB/PCI bridge can claim inbound transactions on the PCI bus for the following command types:

- Memory Read
- Memory Read Line
- Memory Read Multiple
- Memory Write
- Memory Write and Invalidate
- I/O Read
- I/O Write
- Configuration Type 0 Read
- Configuration Type 0 Write

Memory Reads are handled as prefetchable or non-prefetchable, as determined by the address (see *Interrupts and MSI* on page 428 for details). For a region marked non-prefetchable, a corresponding single-beat PLB read is dispatched, which accesses the exact bytes as indicated by the byte enables of the PCI read (discontiguous byte enables are not supported).

For a region marked prefetchable, a corresponding burst PLB read is dispatched with the address truncated to a 128-bit boundary. When the command type is memory read, 32 bytes are read from the PLB. When the command type is memory read line, 64 bytes are read from the PLB, and when the command type is memory read multiple, 256 bytes are read from the PLB.

Note: The PLB/PCI Bridge can be programmed to handle memory read as a memory read line or a memory read multiple (see *Bridge Options Registers* on page 452).

Memory Write and Memory Write and Invalidate are handled exactly the same. They are posted and then written to the PLB. The PLB transfer may include both single-beat and quad-word bursts, depending on the size and alignment of the write.

I/O Read and I/O Write are not handled as connected tenures. Instead, I/O read and I/O write are handled exactly like non-prefetchable memory reads/writes. Thus, I/O reads are delayed reads and I/O writes are posted.

Configuration Reads and Configuration Writes are disconnected after one beat. They access the internal register set directly (no delayed read or write posting). One exception is configuration writes to PCI power management power state that may be handled as a delayed write.

In response to the inbound commands of type Memory or I/O listed above, the PLB/PCI Bridge masters on the PLB the following command types:

- Single Beat Read or Write
- Quadword Fixed Length Burst Read or Write

19.4 Outbound Transaction Handing

The PLB/PCI Bridge can claim outbound transactions on the PLB bus and subsequently initiate a transfer on the PCI bus or access an internal register.

The PLB/PCI Bridge can claim following PLB transactions types:

- Single Beat Read or Write
- 4-Word Line Read or Write
- 8-Word Line Read or Write
- Doubleword Fixed Length Burst Read or Write
- Quadword Fixed Length Burst Read or Write

How these transfers are handled is determined by the address (see *Address Mapping and Address Translation* on page 417 for details). Based on the address, these transaction either access the internal register set or initiate a cycle to the PCI bus.

Based on the PLB address, the PLB/PCI Bridge can initiate the following transactions on the PCI bus:

- Memory Read
- Memory Read Line
- Memory Read Multiple
- Memory Write
- Memory Write and Invalidate
- I/O Read
- I/O Write
- Configuration Read
- Configuration Write
- Special Cycle

Memory Read, Memory Read Line, or Memory Read Multiple is generated for an address to PCI memory space. The Memory Read command is used for transfers of 1 to 4 byte reads. Memory Read Line is used for reads of greater than 4 bytes and less than or equal to 32 bytes (16, 24, or 32 bytes). Memory Read Multiple is used for reads of greater than 32 bytes.

Note: Five to eight byte reads are run as a 32-bit mastered, 2-beat burst, rather than a single 64-bit transfer (this is recommended by the *PCI Local Bus Specification,* Version 2.3).

Memory Write or Memory Write and Invalidate is generated for an address to PCI memory space. The Memory Write and Invalidate is used for transfers beginning and ending on a 64-byte boundary and whose length is a multiple of 64 bytes; otherwise, a Memory Write is used.

I/O Read, I/O Write, and Special Cycle can be generated. The PLB transfer must be single-beat that does not cross a 32-bit boundary.

Configuration Read and Configuration Write, of both Type 0 and Type 1 can be generated. See *External PCI Configuration Cycles* on page 414 for details.

The generation of Interrupt Acknowledge is not supported. Dual address cycles are generated whenever the PCI address is greater than 4GB.

19.5 External PCI Configuration Cycles

PLB masters generate configuration cycles to the PCI bus by accessing the CONFIG_ADDRESS and CONFIG_DATA registers that are located in PLB space. CONFIG_ADDRESS and CONFIG_DATA are implemented as specified in *PCI Local Bus Specification, Version 2.3*, Section 3.2.2.3 (Configuration Mechanism #1).

CONFIG_ADDRESS and CONFIG_DATA must be accessed with single-beat PLB transfers that don't cross a 32-bit boundary.

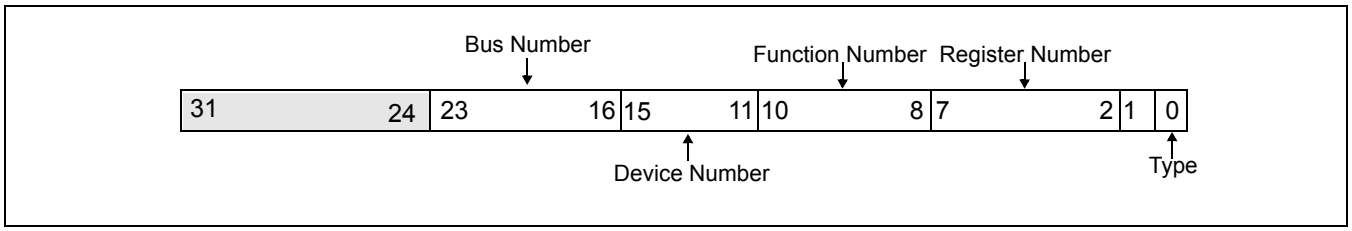
19.5.1 Configuration Mechanism

The general mechanism for accessing PCI configuration space is to write a value into CONFIG_ADDRESS (PCI0_CFGADDR at address 0xC_0EC00000) that specifies the PCI bus number, the device number on that bus, and the configuration register in that device being accessed. Then, a read or write to CONFIG_DATA (PCI0_CFGDATA at address 0xC_0EC00000) causes the bridge to generate the PCI configuration cycle, with the address and type translated from the CONFIG_ADDRESS (PCI0_CFGADDR) value.

19.5.1.1 CONFIG_ADDRESS Register

This register controls what type of cycle is generated when PCI0_CFGDATA is accessed. Its fields are shown in *Figure 19-3*.

Figure 19-3. Format of PCI0_CFGADDR Registers



See *PCI Local Bus Specification, Version 2.3* for details about how the fields are used. Bits 31 to 24 are reserved and always return a 0 when read. Bit 0 determines whether the configuration cycle is Type 0 or Type 1.

19.5.1.2 PCI0_CFGADDR Register Field Description

The bus number, device number, function number, and register number form the address for the register accessed. The type bit field indicates whether the configuration access should pass through a PCI-to-PCI bridge.

The PCI Configuration registers are little endian. When writing PCI0_CFGADDR, use the byte reverse versions of the store word instruction or configure the TLB entry used for PCI register accesses as little endian.

Bus Number (bit 23:16)

0b000000 - Bus 0
0b000001 - Bus 1
0b000010 - Bus 2
.
.
.
0b111101 - Bus 61
0b111110 - Bus 62
0b111111 - Bus 63

Device Number (bits 15:11)
Each Device Number maps to a PCI_AD signal. Up to 16 devices are addressable using PCI_AD16:31. Accessing a device number drives the corresponding PCI_AD signal during a configuration access. The IDSEL of each device on the PCI bus must be connected to a PCI_AD signal so the PCI host can address it with a configuration transaction.

0b00000 - Device 0 attached to PCI_AD16
0b00001 - Device 1 attached to PCI_AD17
0b00010 - Device 2 attached to PCI_AD18
0b00011 - Device 3 attached to PCI_AD19
.
.
.
0b01101 - Device 13 attached to PCI_AD29
0b01110 - Device 14 attached to PCI_AD30
0b01111 - Device 15 attached to PCI_AD31

Function Number (bits 10:8)
0b000 - Function 0
0b001 - Function 1
0b010 - Function 2
0b011 - Function 3
0b100 - Function 4
0b101 - Function 5
0b110 - Function 6
0b111 - Function 7

Register Number (bits 7:2)
The Register Number is the 32-bit word offset within the Configuration Header. Each register within the 32-bit word has a byte offset. The byte offset determines the address within PCI0_CFGDATA that is read or written when accessing a configuration register. The following is the first 64 bytes of the Configuration Header.

0b000000 - Device ID[byte offset 2], Vendor ID[byte offset 0]
0b000001 - Status Register[byte offset 2], Command Register[byte offset 0]
0b000010 - Class Code[byte offset 1], Revision ID[byte offset 0]
0b000011 - BIST[byte offset 3], Header Type[byte offset 2], Latency Timer[byte offset 1], Cache Line Size[byte offset 1]
0b000100 - Base Address 0[byte offset 0]
0b000101 - Base Address 1[byte offset 0]
0b000110 - Base Address 2[byte offset 0]
0b000111 - Base Address 3[byte offset 0]
0b001000 - Base Address 4[byte offset 0]
0b001001 - Base Address 5[byte offset 0]
0b001010 - CardBus CIS Pointer[byte offset 0]
0b001011 - SubSystem ID[byte offset 2], Subsystem Vendor ID[byte offset 0]
0b001100 - Expansion ROM Base Address[byte offset 0]
0b001101 - Reserved[byte offset 1], Capabilities Pointer[byte 0]
0b001110 - Reserved[byte offset 0]
0b001111 - Max_Lat[byte offset 3], Min_GNT[byte offset 2], Interrupt Pin[byte offset 1], Interrupt Line[byte offset 0]

Type (bit 0)
0 - Type 0 access (A configuration access to devices on bus 0)
1 - Type 1 access (A configuration access to devices on 1 or greater)

19.5.1.3 Accessing PCI0_CFGDATA

To access a register within the Configuration Header, write PCI0_CFGADDR with the bus/function/register number/type address and then read or write the appropriate byte offset within the PCI0_CFGDATA register. The byte offset of registers within the Configuration Header is included in Register Name bit field description in *PCI0_CFGADDR Register Field Description* on page 414.

The PCI Configuration registers are little endian. When reading or writing PCI0_CFGDATA, use the byte reverse versions of the load and store instructions or configure the TLB entry used for PCI register accesses as little endian.

PCI0_CFGDATA Byte Offsets:

32-bit Configuration Register
byte offset 0 = 0xC_0EC00004

16-bit Configuration Register
byte offset 0 = 0xC_0EC00004
byte offset 2 = 0xC_0EC00006

8-bit Configuration Register
byte offset 0 = 0xC_0EC00004
byte offset 1 = 0xC_0EC00005
byte offset 2 = 0xC_0EC00006
byte offset 3 = 0xC_0EC00007

Example:

The following example demonstrates how to read the PCI status register from device 4, bus 0, function 0. To access the status register from device 4, bus 0, function 0, 0x00002004 must be stored to PCI0_CFGADDR and byte offset 2 must be read from PCI0_CFGDATA.

```
bit 31:24 = 0b00000000
Bus Number = 0b000000
Device Number = 0b00100
Function Number = 0b000
Register Number = 0b000001
bit 1 = 0
Type = 0


li r4,PCI0_CFGADDR    ! Load r4 with the 32-bit effective address of PCI0_CFGADDR
lis r4,PCI0_CFGADDR
li r5,PCI0_CFGDATA    ! Load r5 with the 32-bit effective address of PCI0_CFGDATA byte offset 2
lis r5,PCI0_CFGDATA   ! Status register is byte offset 2

li r6,0x2004              ! Load r6 with value for PCI0_CFGADDR
lis r6,0x0000
stwbr r4,r6               ! Store byte reverse
lhwbr r7,f5               ! The content of the PCI status word is in r7.
```

Note: The PLB/PCI Bridge will not respond as a PLB slave to accesses to the PCI0_CFGDATA register if the PCI Master Enable bit in the PCI command register is zero because accesses to the PCI0_CFGDATA register require the PLB/PCI Bridge to become a PCI master.

19.6 Address Mapping and Address Translation

The PLB/PCI Bridge register set has several ranges of PLB address space and several ranges of PCI address space that it can claim.

19.6.1 Outbound Address Map

The PLB/PCI Bridge responds as a slave on the PLB in several address ranges. These ranges allow a PLB master to access the internal register set, and to cause the PLB/PCI Bridge to generate memory, I/O, configuration and special cycles to the PCI bus.

PLB masters can access the PLB/PCI Bridge internal register set or PCI slaves in Memory, I/O or Configuration address space by mastering a PLB cycle to an address that indicates the intended destination. There are one or more PLB address ranges for each destination, as shown in *Table 19-2*. If the destination is the internal register set, then the PLB address ranges are fixed.

If the destination is PCI Memory space, there are up to three PLB address ranges that PLB/PCI Bridge can claim. The number of ranges and their location in PLB address space is programmable and determined by the "POM" registers. The PCI address may be a translation of the PLB address. The translation is programmable and also, determined by the POM registers. See *PCI Outbound Map (POM) Configuration* on page 418 for more details.

If the destination is PCI I/O space, PCI Configuration space or PCI special cycle, then the PLB address ranges and the resulting PCI address ranged (translations) are fixed as shown in *Table 19-2* which describes the outbound address map from the view of the PLB, that is, as decoded by the PLB/PCI Bridge as a PLB slave.

Table 19-2. PCI Outbound Address Map

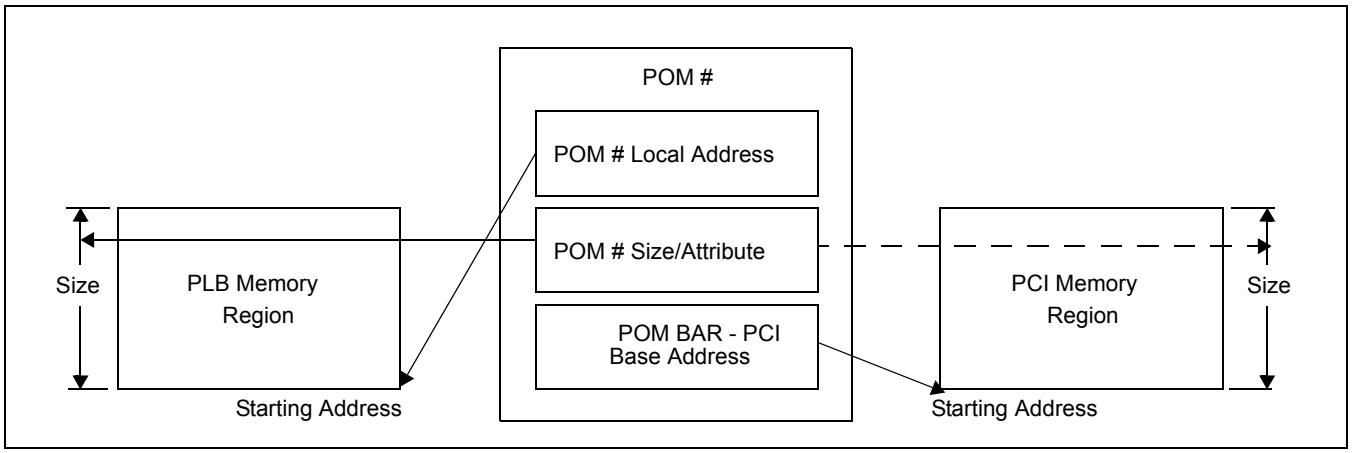
PLB Address Range	Description	PCI Address
0xC 0000 0000 0xC 07FF FFFF	Reserved PCI Bus Controller does not respond	
0xC 0800 0000 0xC 0800 FFFF	PCI I/O Accesses to this range are translated to an I/O access on PCI in the range 0 to 64K-1	0x 0000 0000 0x 0000 FFFF
0xC 0801 0000 0xC 087F FFFF	Reserved, PCI Bus Controller does not respond (Other bridges use this space for discontiguous I/O).	
0xC 0880 0000 0xC 0BFF FFFF	PCI Extra I/O Accesses to this range are translated to an I/O access on PCI in the range 8 MB to 64 MB	0x 0080 0000 0x 03FF FFFF
0xC 0C00 0000 0xC 0EBF FFFF	Reserved PCI Bus Controller does not respond	
0xC 0EC0 0000 0xC 0EC7 FFFF	PCI CONFIG_ADDRESS and CONFIG_DATA 0xC 0EC0 0000: CONFIG_ADDRESS 0xC 0EC0 0004 - 0xC 0EC0 0007: CONFIG_DATA 0xC 0EC0 0008 - 0xC 0EC7 FFFF: Reserved (may contain mirrors of CONFIG_ADDRESS and CONFIG_DATA)	
0xC 0EC8 0000 0xC 0ECF FFFF	PCI Bus Controller registers 0xC 0EC8 0000 - 0xC 0EC8 01FF: internal registers 0xC 0EC8 0200 - 0xC 0ECF FFFF: Reserved (may contain mirrors of internal registers.)	

Table 19-2. PCI Outbound Address Map (Continued)

PLB Address Range	Description	PCI Address
0xC 0ED0 0000 0xC 0EDF FFFF	Special Cycle 0xC 0ED0 0000 4-byte write only: Special Cycle 0xC 0ED0 0000 read only: Reserved 0xC 0ED0 0004 - 0xC FEDF FFFF: Reserved (may contain mirrors of Special Cycle)	
0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF[1]	PCI Memory - Range 0 POM 0 registers map a region in PLB space to a region in PCI memory space. The address ranges are fully programmable.	0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF
0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF[1]	PCI Memory - Range 1 POM 1 registers map a region in PLB space to a region in PCI memory space. The address ranges are fully programmable.	0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF
0x 8000 0000 0000 0000 0x FFFF FFFF FFFF FFFF	PCI Memory - Range 2 POM 2 maps a region in PLB space to a region in PCI memory space. The starting address for Range 2 is fixed at 8000 0000 0000 0000. The PCI address always equals the PLB address minus 8000 0000 0000 0000 in this range.	0x 0000 0000 0000 0000 0x 7FFF FFFF FFFF FFFF
Note 1: These PLB address ranges are software programmable to any value. However, they must be set to values that do not overlap address ranges used by other devices. The PPC460EX/GT address map requires that the POM0 and POM1 PLB address be in the range from 0x 0000 000C 0EE0 0000 to 0x FFFF FFFF FFFF FFFF.		

Figure 19-4 shows the detail of the POM register sets used to map PLB memory regions to PCI address space.

Figure 19-4. POM Register Sets Map PLB Address Space to PCI Address Space



19.6.2 PCI Outbound Map (POM) Configuration

The PLB/PCI Bridge has three ranges in PLB space that are mapped to PCI Memory space. These ranges are referred to as POM 0, POM 1, POM 2. The characteristics of each POM are defined by a set of registers in the PLB/PCI Bridge PCI configuration registers. PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the POM registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

POM 0 is controlled by the following registers:

- POM 0 Local Low Address - PCI0_POM0LAL
- POM 0 Local High Address - PCI0_POM0LAH
- POM 0 Size/Attribute - PCI0_POM0SA
- POM 0 PCI Low Address - PCI0_POM0PCIAL
- POM 0 PCI High Address - PCI0_POM0PCIAH

POM 1 is controlled by the following registers:

- POM 1 Local Low Address - PCI0_POM1LAL
- POM 1 Local High Address - PCI0_POM1LAH
- POM 1 Size/Attribute - PCI0_POM1SA
- POM 1 PCI Low Address - PCI0_POM1PCIAL
- POM 1 PCI High Address - PCI0_POM1PCIAH

POM 2 is controlled by the following register:

- POM 2 Size/Attribute (enable bit only) - PCI0_POM2SA

The location in PLB space of POM0 and POM1 is programmable using the Local Low/High Address registers. The location of POM 2 is fixed as defined above. The PLB address range assigned to each POM should not overlap any other range of PLB space in use. Overlapping will result in undefined behavior.

The range of PCI memory address space that is accessed through POM 0 and POM 1 is also programmable, using the PCI Low Address or PCI High Address registers. This allows address translation between the two busses. The range of PCI memory address space that is accessed through POM 2 is fixed and is a 63-bit address (lower half of the 64-bit memory address space). If the PCI high address is zero, then single-address cycles are generated to the PCI bus. If the PCI high address is greater than zero, then PCI dual address cycles are generated to the PCI bus.

The size of POM 0 and POM 1 is programmable, using the mask portion of the Size/Attribute registers. The size is a power of two, and is a minimum of 1 MB and a maximum of 4 GB. The size of POM 2 is fixed to 2^{63} bytes. The PLB and PCI address spaces for each POM are aligned to this size.

The POMs can be enabled/disabled by the Attribute bits in the Size/Attribute registers. The address ranges should all be initialized before the POM is enabled. See *POM Registers* on page 459 for details on PCI POM registers.

19.6.3 Inbound Address Map

The PLB/PCI Bridge responds as a PCI target for memory accesses to BAR 0 and either BAR2 or Option ROM BAR, and I/O accesses to BAR 1. BAR1 and Option ROM BAR are 32-bit BARs and BAR0 and BAR2 are 64-bit BARs. (The PLB/PCI Bridge also responds as a PCI target for Configuration Type 0 accesses). PCI initiators can access the PLB/PCI Bridge internal register set or PLB slaves by initiating a PCI cycle to an address that indicates the intended destination.

If the destination is the internal register set, then the PCI initiator must run a type 0 configuration cycle, with the appropriate device select, such that the PLB/PCI Bridge's IDSEL is active.

If the destination is PLB space, there are up to two PCI memory address ranges and up to one PCI I/O address range that PLB/PCI Bridge can claim. The number of ranges and their locations in PCI address space is programmable, and determined by the PIM and BAR registers. Also, the PLB address may be a translation of the PCI address. The translation is programmable and is also determined by the PIM registers. See *PCI Inbound Map (PIM) Configuration* on page 420 for more details.

Table 19-3 shows the PCI address maps from the view of PCI, that is, as decoded by the PLB/PCI Bridge as a PCI target.

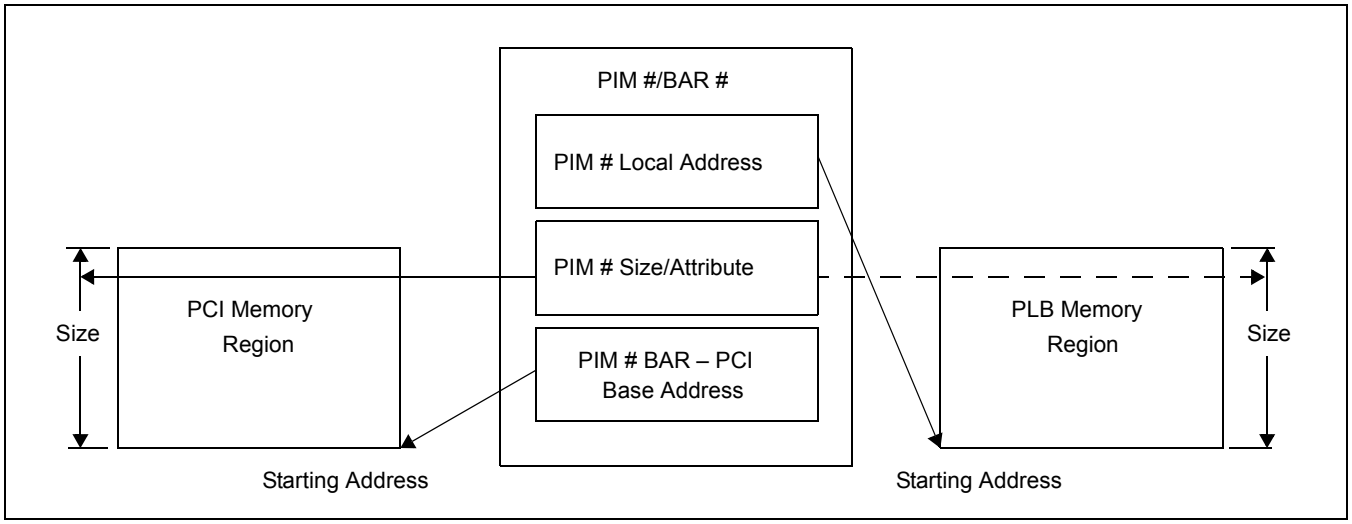
.

Table 19-3. PCI Inbound Address Map

PCI Memory Address	PCI I/O Address	Description	PLB Address
0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF		PCI memory addresses that fall within BAR0 space are normally translated through PIM 0.	0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF
	0x 0000 0000 0x FFFF FFFF	PCI I/O addresses that fall within BAR1 space are translated through PIM 1 to generate the PLB address. The size of PIM 1 is hardcoded to 256 bytes.	0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF
0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF or 0x 0000 0000 FFFF FFFF		PCI memory addresses that fall within BAR2 space or Option ROM BAR Space are translated through PIM 2 to produce a memory address mappable anywhere from 0 to 2^{64} bytes in PLB space.	0x 0000 0000 0000 0000 0x FFFF FFFF FFFF FFFF

Figure 19-5 shows the detail of the PIM/BAR register sets used to map PCI memory regions to PLB address space.

Figure 19-5. PIM Register Sets Map PCI Address Space to PLB Address Space



19.6.4 PCI Inbound Map (PIM) Configuration

The PLB/PCI Bridge has three ranges in PCI space that are mapped to PLB space. These three ranges are referred to as PIM 0, PIM 1 and PIM 2. PIM 0 and PIM 2 map PCI memory space to PLB memory space. PIM 1 maps PCI I/O space to PLB memory space. The characteristics of each PIM are defined by a set of registers in the PLB/PCI Bridge configuration registers. PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

PIM 0 is controlled by the following PCI configuration registers:

- PIM 0 Size/Attribute Register Low - PCI0_PIM0SAL
- PIM 0 Size/Attribute Register High - PCI0_PIM0SAH
- PIM 0 Local Low Address Register - PCI0_PIM0LAL
- PIM 0 Local High Address Register - PCI0_PIM0LAH
- BAR 0 Low Register - PCI0_BAR0L
- BAR 0 High Register - PCI0_BAR0H

PIM 1 is controlled by the following PCI configuration registers:

- PIM 1 Size/Attribute Register - PCI0_PIM1SAL
- PIM 1 Local Low Address Register - PCI0_PIM1LAL
- PIM 1 Local High Address Register - PCI0_PIM1LAH
- BAR 1 Register - PCI0_BAR1L

PIM 2 is controlled by the following PCI configuration registers:

- PIM 2 Size/Attribute Register Low - PCI0_PIM2SAL
- PIM 2 Size/Attribute Register High - PCI0_PIM2SAH
- PIM 2 Local Low Address Register - PCI0_PIM2LAL
- PIM 2 Local High Address Register - PCI0_PIM2LAH
- BAR 2 Low Register - PCI0_BAR2L
- BAR 2 High Register - PCI0_BAR2H

The location in PCI space of each PIM is programmable using the BAR registers.

The range of PLB space that is accessed through each PIM is also programmable. This allows address translation between the two busses. The PLB address is defined in the Local Low/High Address registers.

The size of PIM 0 and PIM 2 is programmable using the Memory Size registers. The size is a power of two, and is a minimum of 4 KB and a maximum of 2^{64} bytes. The PLB and PCI address spaces for each PIM are aligned to this size. The size of PIM 1 is fixed to 256B.

PIM 0, 1 and 2 are enabled or disabled with the enable bit in their size attribute register. PIM 0 and PIM 2 can also be enabled or disabled by the Memory Space bit of the PCI Command Register (PCI0_CMD[MEMA]). PIM 1 can be enabled or disabled by the I/O Space bit of the PCI Command Register (PCI0_CMD[IOA]). The address ranges and sizes of the PIMs should be initialized before they are enabled.

See *PIM Registers* on page 462 for details on PCI PIM registers.

19.7 Data Flow and Buffer Usage

The PLB/PCI Bridge contains dedicated buffers for holding inbound write data (2x256B), inbound read data (1 or 2x256B), outbound write data (2x256B), and outbound read data (1x256B + 1x512B). The buffers are implemented as compilable register arrays. The detailed description of their use is provided as follows.

19.7.1 Inbound Write Post Buffer

There are two 256-byte buffers to handle inbound writes. Each buffer has its own associated address; the second buffer can be filling with a second write while the first buffer is being drained by the PLB master of the PLB/PCI Bridge. The PLB transaction starts when one of the buffers becomes full or the write has completed on the PCI side for that buffer.

The PCI slave of the PLB/PCI Bridge disconnects the PCI transaction when the current buffer is full on 32-byte boundaries. This means that the PCI transaction is disconnected when there is less than 32 bytes remaining in the buffer and a 32-byte address boundary is about to be crossed. This allows the master to continue the write with a memory write with invalidate if the system cache line size is 32 bytes. However, if the system cache line size is greater than 32 bytes, then the master may be forced to continue with a standard memory write transaction.

Note: The buffer space is organized as 128-bit words. Thus, the maximum length that one buffer can accommodate actually depends on the alignment of the starting address. For example, for 256 byte buffers, if the transfer is 128-bit aligned, then 256 bytes can fit. If the address is 128-bit + 1 byte aligned, then only 255 bytes can fit, and so forth. Sizes of 241 bytes or less always fit.

19.7.2 Outbound Write Post Buffer

There are two buffers to handle outbound writes. Each buffer has its own associated address; the second buffer can be filling with a second write while the first buffer is being drained by the PCI master of the PLB/PCI Bridge. The PCI transaction begins when one of the buffers becomes full or when the write has been completed on the PLB side for that buffer.

It is recommended for better performance that PLB masters align their transactions to 64-byte boundaries at the beginning of a transfer. This means that if a master starts a burst that is 64-byte unaligned, it should first run a short transfer that ends at the 64-byte boundary and then start a new transaction that is 64-byte aligned to finish the original transfer. The reason is that the cache line aligned transactions whose total byte length is a multiple of the cache line size are run as memory write with invalidate commands instead of standard memory write transactions. Memory write with invalidate avoids potential snoop push transactions that are unnecessary when entire cache lines are written.

It is not necessary for PLB masters to end their transactions on 64-byte boundaries. The PLB slave of the PLB/PCI Bridge disconnects when the current buffer is full on 64-byte boundaries. This means that a burst write is target disconnected on a 64-byte boundary if the burst write starts 64-byte aligned and the transfer length is not an integer multiple of 64 bytes.

For example, if a (64-byte aligned) 200-byte transfer is being requested, there will be a disconnect at 192 bytes forcing the remaining 8 bytes to be done as a separate transaction. This allows the 192-byte portion of the write to be run on the PCI bus as a memory write with invalidate transaction in conventional PCI mode. If the transfer length is less than or equal to 64-bytes or an integer multiple of 64 bytes, then the entire transfer is taken up to the size of the buffer.

If a PLB write starts unaligned, then the entire byte count is taken up to the size of the buffer, and the transfer is run as a standard PCI memory write. Since transactions that start unaligned cannot be run as memory write with invalidate transactions, they are not target disconnected at 64-byte boundaries. Aligned transactions whose byte length is less than 64-bytes are always run as standard memory write transactions.

19.7.2.1 Outbound Connected Writes

I/O writes, external configuration writes and special cycle writes are considered connected writes and are handled as delayed writes on the PLB. When a buffer is available, these writes are rearbitrated and dispatched to the PCI side. The PLB request is repeatedly rearbitrated until the transfer completes on the PCI side, at which point it is

accepted. While the PLB request is repeatedly rearbitrated, all other outbound reads and writes are rearbitrated except outbound reads that hit in the prefetch buffer that are completed.

19.7.3 Inbound Read Buffer

There is one 256B buffer available to inbound reads in conventional PCI mode. PCI reads are handled as delayed reads. If the delayed read is to non-prefetchable memory, the PCI cycle is disconnected after one beat, regardless of command type (READ, READ LINE, READ MULT), and a single-beat read of the same size is run on the PLB.

If the delayed read is to prefetchable memory, a 32-byte, 64-byte or a 256-byte fixed length burst is run on the PLB to fill the inbound read buffer. A read line transaction fetches 64-bytes. A read multiple transaction fetches 256 bytes. A read transaction can be configured to do a 32-byte, a 64-byte or a 256-byte fetch, based on settings in the PCI target memory read command Interpretation bits in the Bridge Options 1 configuration register. The address for this prefetch read is truncated to 128-bit alignment so that the transaction end is 128-bit aligned.

Only one inbound read is permitted to be in progress at a time, so reads to other addresses are retried until the original read transaction has completed. The delayed read is accepted as soon as there is any data in the buffer. If the PLB side fills as fast or faster than the PCI side drains, then the burst can be sustained. If the PLB side fills slower than the PCI side drains, then the buffer may go empty. When the buffer goes empty, then one wait state is inserted on the PCI to wait for PLB data. If the buffer is still empty after one PCI wait state, then the PCI side is disconnected to allow time for the buffer to fill some more.

After a delayed read completion, data in the PCI inbound read buffer is considered prefetched data. This buffer is treated as a First-In First-Out (FIFO). Random access to the buffer by PCI masters is not supported. As the PCI master reads data from the buffer, the current top of FIFO pointer moves. Data is returned for new reads only if the address of the read transaction exactly matches the current top of FIFO pointer. Otherwise, the read is considered a new read, the FIFO data is discarded, and a new read is transacted to fill the FIFO. A transfer that is bursting out of the buffer is disconnected when it runs out of data in the buffer; there is no automatic refetch based on such a disconnect. The PCI inbound read buffer is marked invalid (flushed) by outbound writes or inbound writes to an address stored in the inbound read buffer as per PCI transaction ordering rules.

Note: Delayed read "hit" is determined by an address match and command of any memory read type (memory read, memory read line, and memory read multiple). The command does not need to exactly match. Also, the byte enables do not need to match, even if the region is non-prefetchable. This is acceptable assuming that no two (or more) different PLB slaves are mapped to the same word of PLB address space (but different bytes).

Note: The 256-byte buffer space is organized as sixteen128-bit words. Thus, the maximum length that one buffer can accommodate actually depends on the alignment of the starting address. If the transfer is 128-bit aligned, then 256 bytes can fit. If the address is 128-bit+1byte aligned, then only 255 bytes can fit, and so forth. Sizes of 241 or less always fit.

19.7.4 Outbound Read Buffer

There is one general purpose buffer and one special purpose buffer used to hold outbound read data. The general purpose buffer and is used by all PLB masters except DMA. One outstanding outbound read at a time is supported.

The special purpose buffer is identified by the PLB master ID and is mapped using the Out Read Special Buffering Mapping register.

The general purpose buffer is a random access prefetch buffer, that is, any read to the general buffer with an address located anywhere within the buffer results in a buffer hit. On buffer hits, data is returned to the PLB master from the buffer and no PCI transaction is initiated.

The processor and other PLB masters have the ability to execute guarded reads. A guarded read has the PLB guarded attribute set.

Guarded reads to the general purpose buffer that are not buffer hits cause the exact amount of data requested (up to 256 bytes) to be fetched. Fixed length bursts of length greater than 256 bytes cause 256 bytes to be fetched. After the data is returned to the PLB master, the buffer is invalidated. Thus, any guarded data that was read from PCI can only be hit one time.

Non-guarded reads to the general purpose buffer that are not buffer hits cause 64 bytes to be fetched if the PLB transfer is for 64 bytes or less, or cause the exact amount (up to 256 bytes) if the PLB transfer is for more than 64 bytes. The data remains in the buffer until some event causes it to be invalidated.

The general purpose buffer is marked invalid (flushed) by writes in either direction or an outbound read that misses this buffer as per PCI transaction ordering rules.

Note: Outbound reads or writes by masters owning special purpose buffers do not invalidate the general purpose buffer.

One PLB master device can be assigned its own dedicated, special purpose read buffer. The PLB Master ID is used to determine which buffer to use for which transaction. In the PPC460EX/GT, the firmware must map the master ID of the DMA controller to this buffer.

When a non-guarded outbound read from this special PLB master occurs, a 512-byte PCI transfer is issued to fill the buffer. Data is returned to the PLB master when the amount of data being requested exists in the buffer. In other words, fixed length PLB read requests are compared to the amount of data available in the buffer associated with the master running the transaction. If there is enough data in the buffer to satisfy the transaction, then data is returned; otherwise, the transaction is retried until the requested number of bytes exist in the buffer. The only exception is if the requested number of bytes extends beyond the length of the end of the buffer so that even when the PCI read completes, there will not be enough data in the buffer to satisfy the request. In this case, the transaction is accepted when the PCI transaction fills to the end of the buffer. Then, the amount of data in the filled-up buffer is returned and the PLB transaction is target disconnected when the end of the buffer is reached.

The special purpose read buffer is treated as a FIFO so that random access to it by PLB masters is not supported. As the PLB master reads data from the buffer, the current top of FIFO pointer moves. Data is returned for new reads from the same master only if the address of the read transaction exactly matches the current top of FIFO pointer. Otherwise, the read is considered a new read; the FIFO data is discarded and a new read is transacted to fill the FIFO.

The guarded attribute controls how data is fetched from the PCI bus. A non-guarded read, when the FIFO is empty, fetches 512-bytes into the buffer, starting with the address of the read. A guarded read, when the FIFO is empty, fetches the length indicated by the PLB master (fixed burst length) from the PCI, up to a maximum of 256 bytes. If the length is greater than 256 bytes, then 256 bytes are fetched. If the FIFO is not empty, a non-guarded read that has a byte count which will completely drain the buffer causes the next sequential 512-byte read to be issued automatically. This read transaction is issued immediately so it can start filling the FIFO as the remaining data is being read out by the PLB master. The PLB read is guaranteed to complete before the read data for the new transaction overwrites it, provided the PLB master does not underflow the read (disconnect prior to completing the requested byte count).

19.7.5 Outbound Connected Reads

I/O reads and external configuration reads are considered connected reads and are handled as delayed reads on the PLB. When a connected read is accepted, it is rearbitrated on the PLB side and dispatched to the PCI side. The PLB request is repeatedly rearbitrated until the transfer completes on the PCI side, at which point, it is accepted. All other outbound reads and writes are rearbitrated during this time.

19.8 Message Passing

This section describes the various message passing methods used by the PCI bridge. The PLB/PCI Bridge supports two methods of message passing: PLB based, which may include I2O messaging unit and simple message passing (built in).

19.8.1 PCI I2O Support

The PLB/PCI Bridge provides some support for *Intelligent I/O Architecture Specification*, Version 1.5 (I2O). This includes the PCI Class Code register and mapping for an I2O Shared Memory space. These features must be used in conjunction with an I2O function (separate from the PLB/PCI Bridge), attached to the PLB.

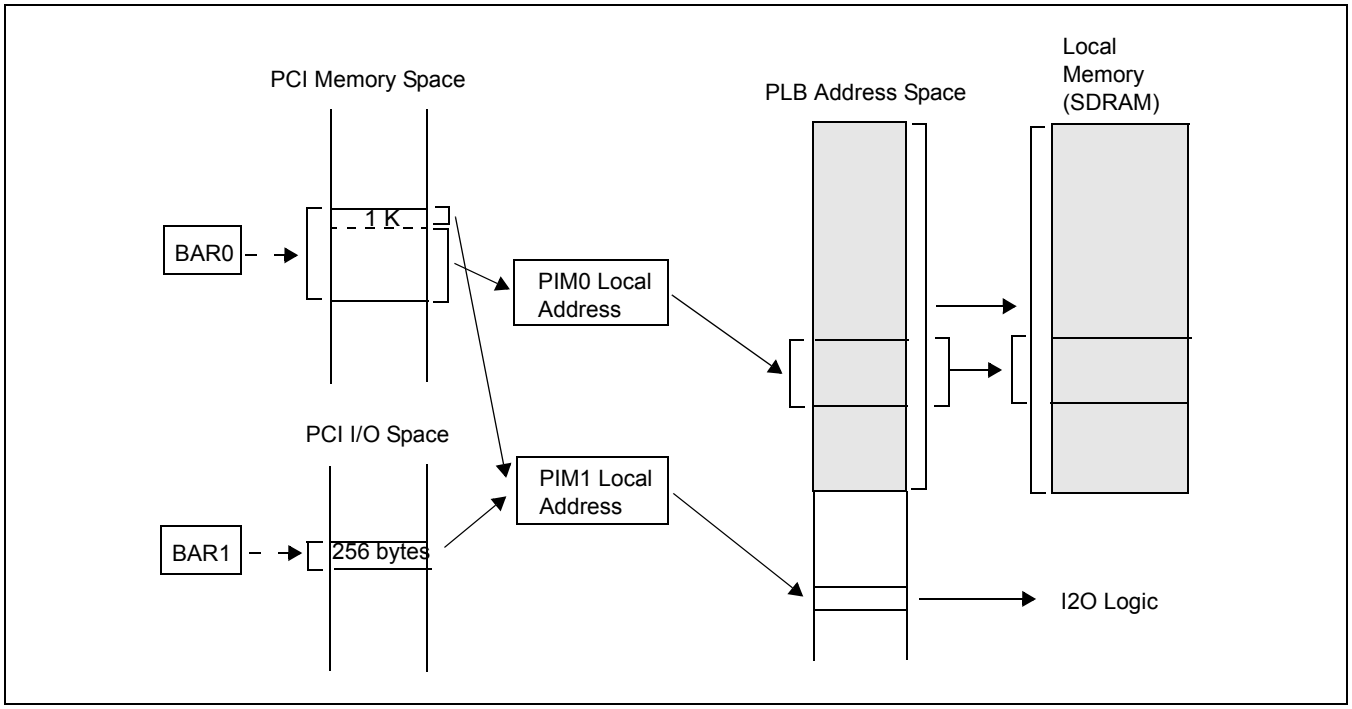
19.8.1.1 PCI Class Code

The PLB/PCI Bridge has the PCI Class Code register in its PCI configuration space. A specific value (0x0E_000X) can be put into this register so that the PPC460EX/GT is identified as I2O capable. See *Intelligent I/O Architecture Specification*, Version 1.5 (I2O).

19.8.1.2 I2O Shared Memory Space

The PLB/PCI Bridge has an I2O shared memory space that maps a portion of PCI memory space to two regions of PLB address space, when the "Split BAR0 Enable" bit and the "Map Split using PIM1" bit of the PIM0 Size/Attribute register are both set. *Figure 19-6* shows the mapping of PCI memory space.

Figure 19-6. Address Mapping for Support of I2O Share Memory Space



The I2O shared memory space is determined by the first memory Base Address Register (BAR 0) in PCI configuration space. See *Intelligent I/O Architecture Specification*, Version 1.5 (I2O).

BAR0 indicates the following:

- Space is to be located in PCI memory space
- Space is to be located in 64-bit address space
- Space is non-prefetchable
- Space has a size of 1MB (changeable by local CPU)
- Starting address (when programmed) of the space within PCI memory space

BAR0 has an associated set of PCI Inbound Mapping (PIM0) registers. The PIM0 registers indicate where in PLB memory space the I2O shared memory space is mapped to. The I2O shared memory space is divided into two regions that are mapped to two different PLB memory spaces.

The first 1 KB of I2O shared memory space is mapped to a 1KB region of PLB memory space, where the I2O Logic resides. The address of this PLB memory space is defined by the PIM1 Local Low/High Address. The I2O Logic includes the following I2O required ports:

- Inbound Free_List and Post_List FIFO port, at offset 0x40
- Outbound Free_List and Post_List FIFO port, at offset 0x44
- I2O Interrupt Status Register, at offset 0x30
- I2O Interrupt Mask Register, at offset 0x34

The remaining offsets (0-3FFh) have miscellaneous uses.

The PIM0 Local Low Address and PIM0 Local High Address registers map the remaining space of the I2O shared memory space (from 1 KB to the end of the region) to a region of PLB memory space, where local memory (SDRAM memory) resides. This region of local memory has been set aside by the local CPU for use as the I2O message frame. When in this mode, regardless of how the PIM0 Size/Attribute, Enable it is set, the 0 to 1KB region does not result in read prefetching on the PLB, and the 1KB and above region does result in read prefetching on the PLB.

Note: The I2O message frame space actually begins in PLB space at PIM0 Local Address + 1KB.

I2O I/O BAR

PCI configuration register BAR1 can be used as an alternate method of locating the I2O Logic in PCI space.

BAR1 indicates the following:

- Space is to be located in PCI I/O space
- Space has a size of 256 bytes
- Starting address (when programmed) of the space within PCI I/O space

BAR1 maps PCI I/O space to PLB address space through PIM1 Local Low/High Address. BAR1 is used by PCI masters that are not capable of accessing the I2O Logic through PCI memory space. Thus, an IOP accesses the I2O Logic through either BAR0 or BAR1, but not both.

Note: Only the first 256 bytes of I2O Logic can be accessed using the I2O I/O BAR, whereas 1 KB can be accessed using the I2O memory BAR.

There is also a BAR2 that maps PCI memory space to PLB address space. However, it is for general purpose use and has no direct association with I2O.

19.8.2 Simple Message Passing Mechanism

For applications that do not use I2O (requiring a separate I2O function on the PLB), a simple message passing mechanism is provided. The simple message passing mechanism enables messages to be exchanged between a PCI master device (the host) and a PLB master device (the local CPU).

The interface consists of a 32-bit Message In register and a 32-bit Message Out register. For 64-bit messages, the Message In High and Message Out High registers may be used.

Note: The message passing registers are 32-bit registers; thus, Message In/Out High cannot be written on the same cycle that Message In/Out (low) is written. Instead, two 32-bit writes are required.

19.8.2.1 Inbound Messages

The Message In register is written with message by the host, and may generate an interrupt to the local CPU. The local CPU can read the message. When processing of the message is completed, the local CPU clears the Message In register by writing to it. The host recognizes completion of the message by polling it (reading it) and waiting for it to be cleared, or by receiving a completion message (using Message Out).

The Message In register generates an interrupt to the local CPU whenever bit 0 of the Message In register is high. Thus, messages that cause an interrupt must have bit 0 set. There are no requirements on any of the other bits. Clearing of the message by the local CPU must include clearing bit 0 and may optionally include changes to other bits. If a 64-bit message is sent, then the host must write the Message In High register before writing the Message In register to ensure the interrupt does not reach the local CPU before Message In High is written.

19.8.2.2 Outbound Messages

The Message Out register is written with message by the local CPU, and may generate an interrupt to the host using the PCI bus. The interrupt may be an Outbound Message Signal Interrupts (MSI). The host can read the message. When processing of the message is completed, the host clears the Message Out register by writing to it. The local CPU recognizes completion of the message by polling it (reading it) and waiting for it to be cleared, or by receiving a completion message (using Message In).

The Message Out register generates an interrupt to the PCI bus whenever bit 0 of the Message Out register is high. Thus, messages that cause an interrupt must have bit 0 set. There are no requirements on any of the other bits. Clearing of the message by the host must include clearing bit 0 and may optionally include changes to other bits. If a 64-bit message is sent, then the local CPU must write the Message Out High register, before writing the Message Out register, to ensure the interrupt doesn't reach the host before Message Out High is written.

Messages sent out, when Outbound MSI is enabled, ensure that any write cycles posted prior to the Message Out are cleared out before the host receives the Message Out interrupt.

19.8.2.3 Access to Messaging Registers

The local CPU accesses the Message In and Message Out registers using a PLB transfer to the register set of the PLB/PCI Bridge, starting at PLB address 0x2_0EC8_0100 (see *PLB/PCI Bridge Registers* on page 439 for more details).

In order for the host to access the Message In and Message Out registers, the "Split BAR0 Enable" bit of the PIM0 Size/Attribute register must be set and the "Map Split using PIM1" bit of the PIM0 Size/Attribute register must be cleared. The host then accesses the Message In and Message Out registers using PCI memory transfers to BAR0 (see *PLB/PCI Bridge Registers* on page 439 for more details).

Note: This mode enables a split in the use of BAR0, as described in *Inbound Interrupt Structure* on page 428, except that the first 1KB of BAR0 is mapped to the Simple Message Passing and Inbound MSI registers, rather than to PLB space.

19.9 Interrupts and MSI

The PLB/PCI Bridge can generate interrupts to the PCI (outbound interrupts) and receive interrupts from the PCI (inbound interrupts). These interrupts may be either standard interrupt signals or Message Signaled Interrupts (MSI). See *PCI Local Bus Specification, Version 2.3*, for details about MSI.

19.9.1 Outbound Interrupt Structure

The Simple Message Passing logic is the source of outbound interrupts and can be routed to the PCI using the PCI_INT(A)# pin, or using MSI. Outbound MSI is usually only used when the PLB/PCI Bridge is in adapter mode.

19.9.2 MSI Capabilities Registers

The PLB/PCI Bridge has a capabilities structure in the PCI configuration space that indicates that it is Outbound MSI capable. The capabilities structure includes the following registers:

- Cap_ID: read only value 05h: indicates the PLB/PCI Bridge is MSI capable.
- Next_Item_Ptr: points to next item in the capabilities list.
- Message Control Register:
  - 64_Bit_Capable: read only value 1.
  - Multiple_Message_Enable: This field indicates how many interrupt messages are allocated to the PLB/PCI Bridge by the system. It will be written by the system to a 0 or 1 indicating one or two messages respectively. The most significant two bits of this field are hardcoded to 0 to prevent errant system software from writing a value greater than 1 to this field.
  - Multiple_Message_Capable: read only value 1, indicating that the PLB/PCI Bridge requests two messages from the system.
  - MSI_Enable: This read/write bit is used by the system to enable or disable MSI capability. A value of 1 means MSIs are enabled.
- Message Address: Contains address bits 0:31 of the address the PLB/PCI Bridge should use to request MSI service. This field is configured by the system.
- Message Upper Address: Contains address bits 63:32 of the address the PLB/PCI Bridge should use to request MSI service. This field is configured by the system.
- Message Data: This field is written by the system. When the PLB/PCI Bridge issues an MSI message, it writes this value to the previously defined message address. When the PLB/PCI Bridge is allocated two messages, the least significant bit of this field determines which message is being reported. If the PLB/PCI Bridge is allocated only one message, then it cannot modify this data on an MSI write.
- Message End of Interrupt: This field is written following an MSI to allow a subsequent MSI to be sent.

19.9.3 Inbound Interrupt Structure

The PLB/PCI Bridge can generate several interrupts to the local (PLB side) interrupt controller (inbound interrupts). The PLB/PCI Bridge has several internal functions that can generate interrupts to the local interrupt controller. There is a unique local interrupt output for Simple Message Passing, PCI Command Register write, PCI Power Management and Error Handling (see *Interrupt Controller Operations* on page 255).

The PLB/PCI Bridge also generates local interrupts based on interrupts received from the PCI bus. This feature is typically used only when the PLB/PCI Bridge is in Host-bridge mode. These interrupts arrive from Inbound MSI and are sent to the local interrupt controller.

When inbound MSI is enabled, one of 12 inbound interrupt signals is driven by the inbound MSI logic and is edge-sensitive (it is active for four PLB clocks). See *Interrupt Controller Operations* on page 255

Inbound MSI is enabled using the Inbound MSI Enable bit in the Bridge Options 1 register. Also, the Split BAR0 Enable bit of PIM0 Size/Attribute must be set. This mode enables a split in the use of BAR0, as described in the Message Passing section, where the first 1 KB of BAR0 is mapped to the Simple Message Passing and Inbound MSI registers, rather than to PLB space.

Note: Inbound MSI cannot be used with a PLB-based I2O function. Typically, Inbound MSI is only used in Host-bridge mode, and I2O function is only used in adapter-bridge mode.

Inbound MSIs are Inbound PCI memory write cycles to address BAR0 + F8h. See *Address Mapping and Address Translation* on page 417 for more details. When a write to the PCI Inbound MSI register occurs, the inbound interrupt that corresponds to the data value is pulsed high.

There is no PCI capabilities structure associated with inbound MSIs.

19.10 PCI Power Management Interface

The PLB/PCI Bridge is designed to the *PCI Bus Power Management Interface Specification,* Version 1.1. This section describes the implementation of this interface in the PLB/PCI Bridge. This includes Capabilities and Power Management Status and Control Registers, Power State Control, and Changing of Power State.

19.10.1 Capabilities and Power Management Status and Control Registers

The PLB/PCI Bridge has a capabilities structure in the PCI configuration space that indicates that the PLB/PCI Bridge is PCI power management capable. The capabilities structure includes the following registers:

- Cap_ID, value 0x01, indicates power management
- Next_Item_Ptr, points to next capabilities structure
- PMC, value 0x0202, indicates no specific capabilities
- PMCSR, hold the current power state
- PMCSR_BSE, value 0x00, unused, not PCI to PLB/PCI Bridge
- Data, value 0x00, not used

See *PLB/PCI Bridge Registers* on page 439 for details.

19.10.2 Power State Control

The current power management state is reported by reading the Power Management and Control Register (PMCSR). The PLB/PCI Bridge supports states D0, D1, D3hot, and D3cold. State D2 is not supported. When the state is not D0, the PLB/PCI Bridge is masked from being a master or a memory or I/O target on the PCI bus. It can still be a configuration target. Thus, accesses that are claimed by the PLB/PCI Bridge when in state D0, are no longer claimed, resulting in master aborts on the PLB or PCI if such an access is attempted.

Note: This mask is independent of the state of the PCI Command Register.

19.10.3 Changing Power States

The PLB/PCI Bridge has two registers that control changing of the power state. The host requests a change in the power state by writing to the Power Management and Control Register (PMCSR). The other register is the PM_SCRR (Power Management State Change Request Register) that provides a method of informing the local processor of a state change request and, thereby, preventing completion of the write to the PMCSR until the local processor indicates it is ready for the state change.

Power state changes are handled as follows:

- If host write to PMCSR requests a change from D3hot to D0, the write is accepted, then the PLB/PCI Bridge causes the entire PPC460EX/GT to be reset.
- All other change requests are handled in the following sequence:
  1. The host requests a new power state by a PCI write to the PMCSR.
  2. The host PCI write is retried unless the PM_SCRR Delayed Write Enable bit is off.
  3. The host PCI write (retried or not) causes the PM_SCRR State Change Request bit to be set that drives an interrupt to the local processor.
  4. The local processor recognizes the interrupt. The local processor can check the PM_SCRR State Change Request bit and the PM_SCRR Requested State bits to determine the nature of the request.
  5. The local processor proceeds to power down those elements on the SOC and outside the SOC (on the adapter) that can be powered down if the requested state is valid.

Note: The PLB/PCI Bridge cannot be powered down or put to sleep based on the PCI power management state.

1. When the subsystem has been powered down, and is ready to change state, the local processor clears the PM_SCRR State Change Request bit and sets the PM_SCRR Accept PMCSR Write bit.
2. When the host PCI write re-occurs, the following takes place:
   1. The host PCI write is accepted.
   2. The PMCSR is updated only if the transition is valid.
   3. PM_SCRR "Accept PMCSR Write" bit is automatically cleared unless the Delayed Write Enable bit is off, in which case the Accept PMCSR Write is always active.
   4. The PLB/PCI Bridge enters the new power state.

Note: Delayed write enable off is a safety mode that can prevent bus hangs if the PM interrupt is not being serviced.

See *PLB/PCI Bridge Registers* on page 439 for more details.

19.10.4 VPD Capability

VPD Capability is added to the capability linked list. The VPD data register, address register and F flag are implemented as described in the *PCI Local Bus Specification, Version 2.3* specification. An edge based PLB side interrupt is generated to the UIC whenever the VPD F-bit is written from the PCI side. This allows the PPC460EX/GT processor to handle the actual reading and writing of the non-volatile memory device. When the host wants to write VPD, it writes the data to the VPD data register. It then writes the address to the VPD address register and writes the F bit to 1. An interrupt is generated causing the PPC460EX/GT processor to read the VPD address and data register, perform the write in the non-volatile memory and then indicate to the PCI side that the write is complete by writing the F bit to 0. When the PCI side sees the F bit become 0, it knows the write has completed.

When the host wants to read VPD, it writes the address to the VPD address register and writes the F bit to 0. An interrupt is generated causing the PPC460EX/GT processor to read the VPD address register and perform the read operation in the non-volatile memory. The PPC460EX/GT processor then writes the read data into the VPD data register and then indicates to the PCI side that the read is complete by writing the F bit to 1. When the PCI side sees the F bit go to 1, it knows the read has completed.

Note that writes from the PLB side to the VPD address field will have no effect. The F bit, itself, can be written from both the PLB and PCI sides.

19.11 PCI Arbiter

The PLB/PCI Bridge includes an optional PCI arbiter that is typically used only in host-bridge mode. This "internal" arbiter can be used with up to four external PCI masters (four $\overline{\text{REQ}}$/$\overline{\text{GNT}}$ pairs) or can be disabled. When disabled, the PLB/PCI Bridge has one $\overline{\text{REQ}}$/$\overline{\text{GNT}}$ pair ($\overline{\text{REQ0}}$/$\overline{\text{GNT0}}$) that attaches to an "external" arbiter. The PCI Arbiter can be enabled either by a boot configuration read from a serial ROM prior to booting (Boot Option G or H) or by software (SDR0_PCI0[PAE]=1).

When using an external PCI/PCI arbiter with the PPC460EX/GT, the internal arbiter is not used, the $\overline{\text{PCI0\_REQ0}}$ and $\overline{\text{PCI0\_GNT0}}$ lines from the PPC460EX/GT are connected to the $\overline{\text{PCI0\_REQ}}$ and $\overline{\text{PCI0\_GNT}}$ lines of the external arbiter. The remaining $\overline{\text{PCI0\_REQ}}$ and $\overline{\text{PCI0\_GNT}}$ lines of the PPC460EX/GT are not used.

The arbiter's algorithm is set as follows:

> The arbiter keeps a set of priorities for each master. If a master loses an arbitration sequence, the losing master's priority increases and the winning master's priority decreases. Fairness is ensured in such a way that the master that keeps losing arbitration cycles eventually has the highest priority and is, therefore, guaranteed to win.

While the PCI bus is busy (not idle), the arbiter grants to the highest priority requester. The arbiter continually recalculates, such that if a new, higher-priority request arrives, the current grant is removed and the new requester receives a grant.

When the PCI bus is idle, the arbiter only recalculates grants if the granted $\overline{\text{REQ}}$ is deasserted or the $\overline{\text{REQ}}$s go from none-active to one (or more) active. Thus, when the PCI bus is idle, the arbiter never removes $\overline{\text{GNT}}$ from a device that has its $\overline{\text{REQ}}$ active.

The arbiter has the following two parking modes:

- Park on PLB/PCI Bridge
- Park on most-recently-active master

The mode is set by the PCI Arbiter Park Mode bit in the Bridge Options 1 (PCI0_BRDGOPT1[PCAR]) register.

Note: If the bus is non-idle, the arbiter always parks on the last granted master, regardless of the parking mode. This is done to avoid unnecessary master latency timeouts.

During reset, the PLB/PCI Bridge High-Zs all PCI outputs, including the $\overline{\text{GNT}}$s.

During reset (both System and PCI), all grant lines are High-Z.

Internal Arbiter Disabled (default for Boot Options A, B, C, D, E and F). SDR0_SDSTP1[PAE] = 0 (bit 13).

- $\overline{\text{PCI0Gnt0Req}}$ = output, request from internal PLB/PCI Bridge for bus ownership.
- $\overline{\text{PCI0Req1:3}}$ = not used.
- $\overline{\text{PCI0Req0Gnt}}$ = input, acknowledge of bus grant to internal PLB/PCI Bridge.
- $\overline{\text{PCI0Gnt1:3}}$ = not used - High-Z.

Internal Arbiter Enabled. SDR0_SDSTP1[PAE] = 1 set by IIC serial ROM bit or register override.

- $\overline{\text{PCI0Req0Gnt}}$ = input request from external PCI master[0].
- $\overline{\text{PCI0Gnt1:3}}$ = input request from external PCI master[1:3].
- $\overline{\text{PCI0Gnt0Req}}$ = output, grant signal to external PCI master[0].
- $\overline{\text{PCI0Gnt1:3}}$ = output, grant signal to external PCI master[1:3].

19.12 Error Handling

The PLB/PCI Bridge controller supports the detection and reporting of several types of errors. The errors are reported to the PLB or the PCI and status information is saved in the configuration register set, so that error type determination can be done.

There are four methods of reporting an error:

- Assert $\overline{\text{PCI0SErr}}$
- Assert $\overline{\text{PCI0PErr}}$
- Generate a PLB MERR (A PLB slave signals the PLB master with Master Read Error, MRdErr, Master Write Error, MWrErr, or a PLB Master Bus Interrupt, MIRQ. MRdErr, MWrErr and MIRQ are error signals on the PLB. PCI0 can generate these errors when it is the PLB slave/PCI Initiator and can receive these errors when it is the PLB master/PCI target.)
- Generate an error interrupt to UIC1 (The PLB/PCI Bridge error interrupt or UIC1 PCI0 Asynchronous Interrupt will be referred to as ERROR_INT from now on).

There are four categories of errors:

- PLB MERR or PLB MIRQ received while a PLB master. This error can be reported to the local CPU using ERROR_INT, which is connected to UIC1 IRQ[25], or to the PCI using $\overline{\text{PCI0SErr}}$.
- PCI error detected or received while a PCI initiator. This error can be reported to the PLB master via PLB MERR. It can also be reported to the local CPU using ERROR_INT or to the PCI using $\overline{\text{PCI0SErr}}$.
- PCI error detected while a PCI slave. This error can be reported to the PCI initiator using $\overline{\text{PCI0PErr}}$. It can also be reported to the local CPU using ERROR_INT or to the PCI using $\overline{\text{PCI0SErr}}$.
- Asynchronous error detected. This error can be reported to the local CPU using ERROR_INT or to the PCI using $\overline{\text{PCI0SErr}}$.

Each error that can be detected has a mask associated with it. If the mask is set, then the detection of that error condition is disabled. There are also separate masks for the $\overline{\text{PCI0SErr}}$, $\overline{\text{PCI0PErr}}$, PLB MERR, and ERROR_INT, that prevent reporting of these types of errors. These masks do not prevent the detection of the errors.

19.12.1 Error Types

The following are the error types:

- While a PLB master, receive PLB Read MERR or PLB Write MERR or PLB MIRQ.
- While a PCI Initiator, receive master abort.
- While a PCI Initiator, receive target abort.
- While a PCI Initiator, receive/detect a data parity error.
- While a PCI target, detect a data parity error.
- While a PCI target, detect an address or attribute parity error.
- Async to busses, detect delayed read discard timer expired.
- Async to busses, receive $\overline{\text{PCI0SErr}}$.

19.12.2 Error Descriptions

The following subsections describe in detail how these errors are handled, what actions are taken for each error, and how to reset a given error.

19.12.2.1 While a PLB Master, Receive PLB Read MERR or PLB Write MERR or PLB MIRQ

This error occurs when the PLB/PCI Bridge runs an inbound transfer to the PLB, but the PLB slave asserts PLB Read MERR or PLB Write MERR or PLB MIRQ (referred to collectively as PLB MERR). The PLB/PCI Bridge does not attempt to associate the PLB MERR with a specific PLB cycle.

If this error is detected, the PLB/PCI Bridge still completes the transfer on the PLB and PCI busses.

The following register bits are involved in this error:

- If the MErr Receive Enable bit of the Error Enable register (PCI0_ERREN[RERE, WERE, MQRE]) is set, then detection of this error is enabled. If this error occurs, then the following registers are updated:
  - The MErr Received bit of the Error Status register (PCI0_ERRSTS[MRER, MWER, MQR]) is set.
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture superfluous data.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures superfluous data.

  These register hold their values, regardless of the detection of other errors, until the Error Status register (PCI0_ERRSTS) is cleared.
- This error can be reported locally or to the PCI, as determined by the Route Error Locally bit of the Error Enable register (PCI0_ERREN[REL]). If PLB MERR is detected as an error and routed locally (PCI0_ERREN[REL]), and the ERROR_INT Assertion Enable bit of the Error Enable register is set (PCI0_ERREN[EAE]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status register (PCI0_ERRSTS[EIS]) is set.
- If PLB $\overline{\text{MERR}}$ is detected as an error (PCI0_CMD[SERE]=1) and routed to the PCI (PCI0_ERREN[REL]=0), and the $\overline{\text{SERR}}$ Enable bit of the PCI Command register is set (PCI0_ERREN[SEE]=1), then $\overline{\text{PCI0SErr}}$ is asserted, and the Signaled System Error bit of the PCI Status register (PCI0_STATUS[SSE]=1) is asserted.

19.12.2.2 While a PCI Initiator, Receive Master Abort

This error is detected when the PLB/PCI Bridge runs an outbound transfer to the PCI bus, but no target responds with PCI_DevSel within the required time-out window and error detection is enabled.

Note: An error is never detected if the master abort occurs on a special cycle or a configuration cycle.

If this error is detected on a read, the PLB/PCI Bridge still completes the transfer on the PLB, but drives all ones on the read data bus. On writes, the data is discarded.

The following register bits are involved in this error:

- If the Master Abort Error Enable bit of the Error Enable Register is set (PCI0_ERREN[MAEE]=1), then detection of this error is enabled. If this error occurs, the following registers are updated:
  - The Master-Abort bit of the PCI Status Register is set (PCI0_STATUS[RMA]=1).
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture the PLB address.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures information about the PLB address.

  These registers hold their values, regardless of the detection of other errors, until the PCI Status Register (PCI0_STATUS) is cleared.
- On reads and connected-tenure writes, if the MERR Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[MRWE]=1), then PLB read or write MERR is asserted on the failed data beat, and all subsequent data beats, and the MERR Signaled bit of the Error Status Register is set (PCI0_ERRSTS[MRWE]=1).
- This error can be reported locally or to the PCI (PCI0_ERREN[REL]=0), as determined by the Route Error Locally bit of the Error Enable Register (PCI0_ERREN[REL]). If master abort is detected (PCI0_ERREN[MAEE]=1) as an error and routed locally (PCI0_ERREN[REL]=1) and the ERROR_INT Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[EAE]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status Register is set (PCI0_ERRSTS[EIS]=1).
- If master abort is detected as an error (PCI0_ERREN[MAEE]=1) and routed to the PCI, and the SERR Enable bit of the PCI Command Register is set (PCI0_CMD[SERE]=1), then PCI0SErr is asserted, and the Signaled System Error bit of the PCI Status Register (PCI0_STATUS[SSE]=1) is asserted.

19.12.2.3 While a PCI Initiator, Receive Target Abort

This error is detected when the PLB/PCI Bridge runs an outbound transfer to the PCI bus and the PCI target signals a target abort.

If this error is detected on a read, the PLB/PCI Bridge still completes the transfer on the PLB, but the read data bus is all ones. On a write, the data is discarded.

Note: If target abort is received on a read, any data received before the target abort is put in the read buffer and may be passed to the PLB master.

The following register bits are involved in this error:

- If the Target Abort Error Enable bit of the Error Enable Register is set (PCI0_ERREN[TAEE]=1), then detection of this error is enabled. If this error occurs, the following registers are updated:
  - The Target-Abort bit of the PCI Status Register is set (PCI0_STATUS[RTA]=1).
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture the PLB address.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures information about the PLB address.

  These registers hold their values, regardless of the detection of other errors, until the Error Status Register (PCI0_ERRSTS) is cleared.

- On reads and connected-tenure writes, if the MERR Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[MRWE]=1), then PLB read or write MERR is asserted on the failed data beat, and all subsequent data beats and the MERR Signaled bit of the Error Status Register (PCI0_ERRSTS[MRWE]) is set.
- This error can be reported locally or to the PCI, as determined by the Route Error Locally bit of the Error Enable Register (PCI0_ERREN[REL]). If target abort is detected as an error and routed locally (PCI0_ERREN[REL]=1), and the ERROR_INT Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[EAE]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status Register is set (PCI0_ERESTS[EIS]=1).
- If target abort is detected as an error (PCI0_ERREN[TAEE]=1) and routed to the PCI (PCI0_ERREN[REL]=0), and the SERR Enable bit of the PCI Command Register is set (PCI0_CMD[SERE]=1), then PCI0SErr is asserted, and the Signaled System Error bit of the PCI Status Register (PCI0_STATUS[SSE]=1) is asserted.

19.12.2.4 While a PCI Initiator, Receive/Detect Data Parity Error

This error is detected when the PLB/PCI Bridge runs an outbound transfer to the PCI bus and a parity error is detected on read data or PCI0PErr is received during a write.

If this error is detected on a read, the PLB/PCI Bridge still completes the transfer on the PLB and PCI busses.

The following register bits are involved in this error:

- If a data parity error is detected on a read, the Detected Parity Error bit of the PCI Status Register is set (PCI0_STATUS[DPE]=1). Setting of this bit is non-maskable. It can be reset by writing a 1 to it.
- If the Parity Error Response bit of the PCI Command Register is set (PCI0_CMD[PER]=1), then detection of this error is enabled. If this error occurs, the following registers are updated:
  - The Master Data Parity Error bit of the PCI Status Register is set (PCI0_STATUS[DPE]=1).
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture the PLB address.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures info about the PLB address.

  These registers hold their values, regardless of the detection of other errors, until the PCI Status Register (PCI0_STATUS) is cleared.
- On reads, PCI0PErr is asserted.
- On reads and connected-tenure writes, if the MERR Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[MRWE]=1), then PLB read or write MERR is asserted on the failed data beat, and all subsequent data beats, and the MERR Signaled bit of the Error Status Register is set (PCI0_ERRSTS[MRWE]=1). If the parity error is detected on the last one or two beats of a read and the PCI frequency is low, it may fail to be reported using MErr.
- This error can be reported locally or to the PCI, as determined by the Route Error Locally bit of the Error Enable Register (PCI0_ERREN[REL]). If a master data parity error is detected as an error and routed locally (PCI0_ERREN[REL]=1), and the ERROR_INT Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[EAE]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status Register is set (PCI0_ERRSTS[EIS]=1).
- If this error is detected and routed to the PCI (PCI0_ERREN[REL]=0), and the SERR Enable bit of the PCI Command Register is set (PCI0_CMD[SERE]=1), then PCI0SErr is asserted, and the Signaled System Error bit of the PCI Status Register (PCI0_STATUS[SSE]=1) is asserted.

19.12.2.5 While a PCI Target, Detect Data Parity Error

This error is detected when the PLB/PCI Bridge receives an inbound write from the PCI bus and a parity error is detected on the write data.

If this error is detected, the PLB/PCI Bridge still completes the transfer on the PLB and PCI busses.

The following register bits are involved in this error:

- If a data parity error is detected, the Detected Parity Error bit of the PCI Status Register is set (PCI0_STATUS[DEPE]=1). Setting of this bit is maskable by PCI0_ERREN[PCIPO]. It can be reset by writing a 1 to it.
- If the Parity Error Response bit of the PCI Command Register is set (PCI0_CMD[PER]=1), $\overline{\text{PCI0PErr}}$ is asserted.
- If both the Parity Error Response bits of the PCI Command Register (PCI0_CMD[PER]=1), the Inbound Write Data Parity Enable bit (PCI0_ERREN[IDPE]=1), and the Error Enable Register are set, then detection of this error is enabled. If this error occurs, the following registers are updated:
  - The Inbound Write Data Parity Error Detected bit of the Error Status Register is set (PCI0_STATUS[DEPE]=1).
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture superfluous data.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures superfluous data.

  These registers hold their values, regardless of the detection of other errors, until the PCI Status Register (PCI0_STATUS) is cleared.
- This error can be reported locally or to the PCI, as determined by the Route Error Locally bit of the Error Enable Register (PCI0_ERREN[REL]). If a master data parity error is detected as an error and routed locally (PCI0_ERREN[REL]=1), and the ERROR_INT Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[EAE]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status Register is set (PCI0_ERRSTS[EIS]=1).
- If this error is detected and routed to the PCI (PCI0_ERREN[REL]=0), and the $\overline{\text{SERR}}$ Enable bit of the PCI Command Register is set (PCI0_CMD[SERE]=1), then $\overline{\text{PCI0SErr}}$ is asserted, and the Signaled System Error bit of the PCI Status Register (PCI0_STATUS[SSE]=1) is asserted.

19.12.2.6 While a PCI Target, Detect Address or Attribute Parity Error

This error is detected when the PLB/PCI Bridge receives an inbound transfer from the PCI bus and a parity error is detected on the address or the attribute (PCI only).

The PCI transfer is completed normally (as if no error). The PLB transfer is completed normally (as if no error).

The following register bits are involved in this error:

- If an address or attribute parity error is detected, the Detected Parity Error bit of the PCI Status Register is set (PCI0_STATUS[DEPE]=1). Setting of this bit is non-maskable. It can be reset by writing a 1 to it.
- If the Parity Error Response bit of the PCI Command Register (PCI0_CMD[PER]=1) and the Address/Attribute Parity Error Enable bit of the Error Enable Register are set (PCI0_ERREN[APEE]=1), then detection of this error is enabled. If this error occurs, the following registers are updated:
  - The Address/Attribute Parity Error Detected bit of the Error Status Register is set (PCI0_ERRSTS[APED]=1).
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture superfluous data.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures superfluous data.

These registers hold their values, regardless of the detection of other errors, until the Error Status Register (PCI0_ERRSTS) is cleared.

- This error can be reported locally or to the PCI, as determined by the Route Error Locally bit of the Error Enable Register (PCI0_ERREN[REL]). If a master data parity error is detected as an error and routed locally (PCI0_ERREN[REL]=1), and the ERROR_INT Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[EAE]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status Register is set (PCI0_ERRSTS[EIS]=1).
- If this error is detected and routed to the PCI (PCI0_ERREN[REL]=0), and the $\overline{\text{SERR}}$ Enable bit of the PCI Command Register is set (PCI0_CMD[SERE]=1), then $\overline{\text{PCI0SErr}}$ is asserted, and the Signaled System Error bit of the PCI Status Register is (PCI0_STATUS[SSE]=1) asserted.

19.12.2.7 Detect Delayed Read Discard Timer Expired

This error is detected when the PLB/PCI Bridge runs an inbound read as a delayed read, but the PCI Initiator never returns for the data, causing the Discard Timer to expire.

This error is detected asynchronous to any bus activity.

The following register bits are involved in this error:

- If the Delayed Read Discard Timer Expired Error Enable bit of the Error Enable Register is set (PCI0_ERREN[DRTE]=1), then detection of this error is enabled. If this error occurs, the following registers are updated:
  - The Delayed Read Discard Timer Expired bit of the Error Status Register is set (PCI0_ERRSTS[DTE]=1).
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture superfluous data.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures superfluous data.

  These registers hold their values, regardless of the detection of other errors, until the Error Status Register (PCI0_ERRSTS) is cleared.
- This error can be reported locally or to the PCI, as determined by the Route Error Locally bit of the Error Enable Register (PCI0_ERREN[REL]). If this error is detected and routed locally, and the ERROR_INT Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[REL]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status Register is set (PCI0_ERRSTS[EIS]=1).
- If this error is detected and routed to the PCI (PCI0_ERREN[REL]=0), and the $\overline{\text{SERR}}$ Enable bit of the PCI Command Register is set (PCI0_CMD[SERE]=1), then $\overline{\text{PCI0SErr}}$ is asserted, and the Signaled System Error bit of the PCI Status Register (PCI0_STATUS[SSE]=1) is asserted.

19.12.2.8 Received $\overline{\text{PCI0SErr}}$

This error is detected whenever $\overline{\text{PCI0SErr}}$ is asserted. This allows the local CPU to be the handler for $\overline{\text{PCI0SErr}}$.

The following register bits are involved in this error:

- If the $\overline{\text{PCI0SErr}}$ Received as Error Enable bit of the Error Enable Register is set (PCI0_ERREN[SERE]=1), then detection of this error is enabled. If this error occurs, the following registers are updated:
  - The $\overline{\text{PCI0SErr}}$ Received bit of the Error Status Register is set (PCI0_ERREN[PSEE]=1).
  - The PLB Slave Error Address Low and High registers (PCI0_PLBEARL/H) capture superfluous data.
  - The PLB Slave Error Attribute Register (PCI0_PLBBESR) captures superfluous data.

  These registers hold their values, regardless of the detection of other errors, until the Error Status Register (PCI0_ERRSTS) is cleared.

- This error is always reported locally regardless of the state of the Route Error Locally bit of the Error Enable Register (PCI0_ERREN[REL]). If this error is detected, and the ERROR_INT Assertion Enable bit of the Error Enable Register is set (PCI0_ERREN[EAE]=1), then ERROR_INT is asserted, and the ERROR_INT Signaled bit of the Error Status Register is set (PCI0_ERRSTS[EIS]).

19.12.3 Read Prefetch Boundary Crossing

In general, it is the responsibility of PLB masters and PCI Initiators not to cross any physical boundaries of the slave devices they are accessing. However, on read cycles, the PLB/PCI Bridge is capable of prefetching. Since the PLB/PCI Bridge has no knowledge of the slave devices it is accessing, it is possible for the PLB/PCI Bridge to generate a prefetching read that does cross a physical boundary of the slave device.

The PLB/PCI Bridge and the PLB and PCI busses are designed such that read prefetch boundary crossing does not cause errors.

19.12.3.1 Inbound Read Prefetch Boundary Crossing Handling

The PLB/PCI Bridge disconnects inbound reads on the PCI bus at the boundary of its address space, as defined by the BAR registers. In response to inbound read requests, the PLB/PCI Bridge may request a PLB prefetching read that may cross any boundary. It is the responsibility of the PLB slave to disconnect if the burst reaches one of its boundaries. The PLB/PCI Bridge does not attempt to complete a read that is disconnect by the PLB slave. This ensures that the PLB/PCI Bridge never prefetches into an invalid PLB address when it starts reading from a valid PLB address.

19.12.3.2 Outbound Address Space Boundary Handling

The PLB/PCI Bridge selects an amount to read from the PCI bus that may include some prefetching. If the PCI bus is disconnected, the PLB/PCI Bridge resumes the read until completion only if the disconnect is not on a 4KB boundary. Since PCI memory devices are aligned to 4KB boundaries, this prevents prefetching into another devices memory space. It is the responsibility of the PCI Slave to disconnect, if a burst reaches its boundary.

19.13 Oddities

This section describes items of the PLB/PCI Bridge which may be different from expected.

19.13.1 Discontiguous and Irregular Byte Enable Handling

Outbound transactions cannot generate discontiguous byte enables because they are unsupported on the PLB.

The PLB/PCI Bridge does not support discontiguous byte enables for inbound transactions, that is, byte enables that are inactive in between active byte enables. An inbound transaction with discontiguous byte enables completes normally on the PCI bus, and no errors are detected. However, the results are undefined (which bytes are actually accessed is not determined).

Inbound reads to a non-prefetchable region (always single beat) with no active byte enables result in a 1-B read on the PLB.

19.13.2 PLB Arbiter Must Be Fair

In order to ensure that a PLB master that has a delayed read pending has a chance to receive its data, the PLB arbiter must be in FAIR mode and all PLB masters that access the PCI must drive the same PLB priority level as the PLB/PCI Bridge.

19.14 PLB/PCI Bridge Registers

The PLB/PCI Bridge register set consists of internal registers used for controlling the PLB/PCI Bridge. These registers can be accessed from both the PLB and the PCI (if enabled). Most are accessed from the PCI by Configuration Type 0 cycles when the IDSEL input of PLB/PCI Bridge is active. The Message Passing and Inbound MSI registers are accessed from the PCI by memory cycles to the address pointed to by the PCI BAR0 Register.

PLB masters access these registers directly in the 0xC_0EC8_0000 to 0xC_0EC8_01FF address range.

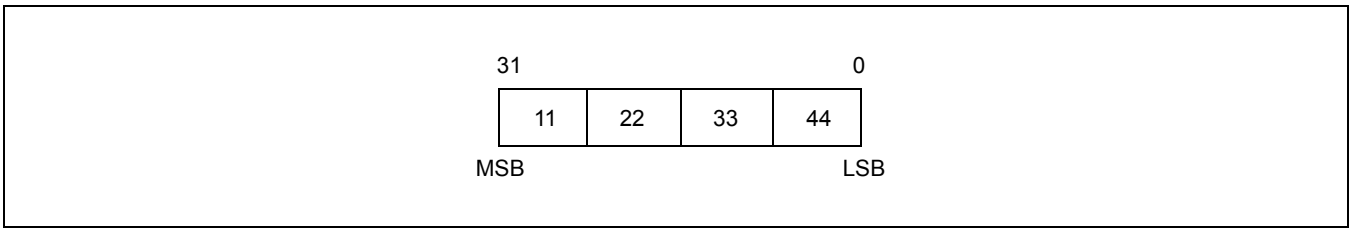
These registers must be accessed with single-beat, one to four byte transfers. Software must use appropriate masks to extract the desired bits when the register includes reserved bits. Writes must preserve the values of reserved bit positions by first reading the register, merging the new value, and writing the result.

19.14.1 Byte Ordering

The register set of the PLB/PCI Bridge is described using little endian bit ordering. Thus, the least significant bit is bit 0 and is always shown on the right.

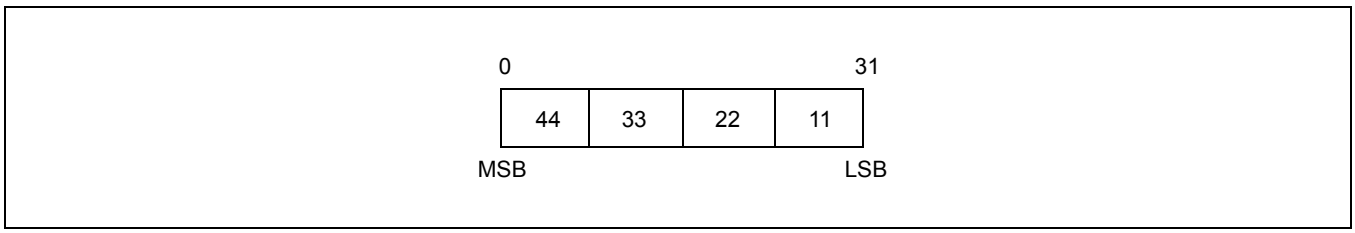
Furthermore, the PLB/PCI Bridge hardware implements the registers in little endian byte ordering. Thus, software that runs in big endian mode must take this into account when accessing the registers. For example, as shown in *Figure 19-7*, it is desirable to put a value in the 32-bit register.

Figure 19-7. Little Endian



Then, big endian software should either perform a store (STW) of the value 44332211 or do a byte-reversed store (STWBR) of the value 11223344. If the microprocessor supports little endian memory pages, then software can perform a store (STW) of the value 11223344 to a little endian page that maps to the configuration registers. In all these cases, *Figure 19-8* shows what will appear on the appropriate word of the PLB data bus.

Figure 19-8. PLB Data Bus Value



19.14.2 PPC460EX/GT Chip Control Registers

The following three registers described under this section are system DCR registers which are accessed indirectly through the SDR0_CFGADDR and SDR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions.

19.14.2.1 PCI Control Register 0 (SDR0_PCI0)

SDR0_PCI0 contains PCI control information. It is configured by bootstraps.

Figure 19-9. PCI Configuration Register (SDR0_PCI0)

0	PAE	PCI Arbiter Enable: 0 PCI arbiter disabled 1 PCI arbiter enabled	Reset value = SDR0_SDSTP1[PAE]
1	PHCE	Enable PCI to be Configured by External Host: 0 PCI host configuration disabled 1 PCI host configuration enabled	Reset value = SDR0_SDSTP1[PHCE]
2	PISE	PCI Initial Sequence Mode Definition: 0 Adapter mode 1 Host mode	Reset value = SDR0_SDSTP1[PISE]
3	PCWE	PCI Local CPU Wait Enable: 0 PCI local CPU wait disabled 1 PCI local CPU wait enabled	Reset value = SDR0_SDSTP1[PCWE]
4:7	PPIM	PCI Inbound Map (PIM) Settings 0000 PIM0 off, PIM1 off, PIM2 off 0001 PIM0 4K, PIM1 off, PIM2 off 0010 PIM0 1M, PIM1 off, PIM2 off 0011 PIM0 64M, PIM1 off, PIM2 off 0100 PIM0 4K prefetch enabled, PIM1 off, PIM2 off 0101 PIM0 1M prefetch enabled, PIM1 off, PIM2 off 0110 PIM0 64M prefetch enabled, PIM1 off, PIM2 off 0111 PIM0 64K, PIM1 off, PIM2 16k 1000 PIM0 1M, PIM1 off, PIM2 64k 1001 PIM0 64K prefetch enabled, PIM1 off, PIM2 16K 1010 PIM0 1M prefetch enabled, PIM1 off, PIM2 64K 1011 PIM0 64K, PIM1 off, PIM2 64K prefetch enabled 1100 PIM0 1M, PIM1 off, PIM2 1Mp 1101 PIM0 1M prefetch enabled, PIM1 off, PIM2 1M prefetch enabled 1110 PIM0 1M, PIM1 on, PIM2 off 1111 PIM0 1M, PIM1 on, PIM2 16K	These bits control the default settings for various PIM fields. Reset value = SDR0_SDSTP1[PPIM]
8:31		Reserved	

19.14.3 PLB PCI Register Categories

All the PLB/PCI registers described in this chapter are categorized as follows:

- *PCI Standard Header Registers* on page 444
- *Bridge Options Registers* on page 452
- *Error Handling Registers* on page 456
- *POM Registers* on page 459
- *PIM Registers* on page 462
- *MSI Capability Block Definition Registers* on page 468
- *Power Management Register Block Definition Registers* on page 470
- *PCI Capability Block Definition Registers* on page 473
- *Vital Product Data (VPD) Registers* on page 475
- *Simple Message Passing and Inbound MSI Registers* on page 476

The two registers listed in *Table 19-4* are used to access the configuration registers of external PCI devices only. See *Configuration Mechanism* on page 414 for a description of register accesses using the PCI0_CFGADDR and PCI0_CFGDATA registers.

Table 19-4. External PCI Device Configuration Register Access

Mnemonic	Register	Address	Access
PCI0_CFGADDR	PCI Configuration Address Register	0xC 0EC00000	R/W
PCI0_CFGDATA	PCI Configuration Data Register	0xC 0EC00004	R/W

Registers listed in *Table 19-5* can be accessed in two ways:

1. By the processor using the PLB address
2. By an external PCI device using configuration cycles to the proper offset.

Table 19-5. Internal PCI Configuration Registers

Mnemonic	PLB		PCI Configuration		Register	Page
	Address	Access	Offset	Access		
PCI0_VENDID	0xC 0EC80000	R/W	0x01 – 0x00	R	PCI Vendor ID	444
PCI0_DEVID	0xC 0EC80002	R/W	0x03 – 0x02	R	PCI Device ID	444
PCI0_CMD	0xC 0EC80004	R/W	0x05 – 0x04	R/W	PCI Command Register	444
PCI0_STATUS	0xC 0EC80006	R/W	0x07 – 0x06	R/W	PCI Status Register	445
PCI0_REVID	0xC 0EC80008	R/W	0x08	R	PCI Revision ID	446
PCI0_CLS	0xC 0EC80009	R/W	0x0B – 0x09	R	PCI Class Register	446
PCI0_CACHELS	0xC 0EC8000C	R/W	0x0C	R/W	PCI Cache Line Size	447
PCI0_LATTIM	0xC 0EC8000D	R/W	0x0D	R/W	PCI Latency Timer	447
PCI0_HDTYPE	0xC 0EC8000E	R	0x0E	R	PCI Header Type	447
PCI0_BIST	0xC 0EC8000F	R	0x0F	R	PCI Built In Self Test Control	447
PCI0_ BAR0L	0xC 0EC80010	R/W	0x13 – 0x10	R/W	PCI BAR 0 Low	448
PCI0_ BAR0H	0xC 0EC80014	R/W	0x17 – 0x14	R/W	PCI BAR 0 High	448
PCI0_BAR1	0xC 0EC80018	R/W	0x1B – 0x18	R/W	PCI BAR 1	449
PCI0_ BAR2L	0xC 0EC8001C	R/W	0x1F – 0x1C	R/W	PCI BAR 2 Low	449
PCI0_ BAR2H	0xC 0EC80020	R/W	0x23 – 0x20	R/W	PCI BAR 2 High	450
PCI0_SBSYSVID	0xC 0EC8002C	R/W	0x2D – 0x2C	R	PCI Subsystem Vendor ID	450
PCI0_SBSYSID	0xC 0EC8002E	R/W	0x2F – 0x2E	R	PCI Subsystem ID	450
PCI0_EROMBA	0xC 0EC80030	R/W	0x33 – 0x30	R/W	PCI Expansion ROM Base Address	451
PCI0_CAP	0xC 0EC80034	R/W	0x34	R	PCI Capabilities Pointer	451
PCI0_INTLN	0xC 0EC8003C	R/W	0x3C	R/W	PCI Interrupt Line	451
PCI0_INTPN	0xC 0EC8003D	R	0x3D	R	PCI Interrupt Pin	452
PCI0_MINGNT	0xC 0EC8003E	R/W	0x3E	R	PCI Minimum Grant	452
PCI0_MAXLTNCY	0xC 0EC8003F	R/W	0x3F	R	PCI Maximum Latency	452

Table 19-5. Internal PCI Configuration Registers (Continued)

Mnemonic	PLB		PCI Configuration		Register	Page
	Address	Access	Offset	Access		
PCI0_BRDGOPT1	0xC 0EC80040	R/W	0x43 – 0x40	R/W	PCI Bridge Options 1	452
PCI0_BRDGOPT2	0xC 0EC80044	R/W	0x47 – 0x44	R/W	PCI Bridge Options 2	454
PCI0_VPDCAPID	0xC 0EC80048	R	0x48	R	PCI VPD Capability Identifier	475
PCI0_VPDNIPTR	0xC 0EC80049	R/W	0x49	R	PCI VPD Next Item Pointer	475
PCI0_VPDADR	0xC 0EC8004A	R/W	0x4A	R/W	PCI VPD Address	475
PCI0_VPDDATA	0xC 0EC8004C	R/W	0x4C	R/W	PCI VPD Data	476
PCI0_ERREN	0xC 0EC80050	R/W	0x53 – 0x50	R/W	PCI Error Enable	456
PCI0_ERRSTS	0xC 0EC80054	R/W	0x57 – 0x54	R/W	PCI Error Status	457
PCI0_PLBBESR	0xC 0EC80058	R	0x5B – 0x58	R	PCI PLB Slave Error Syndrome Register	458
PCI0_PLBBEARL	0xC 0EC8005C	R	0x5F – 0x5C	R	PCI PLB Slave Error Address Register Low	458
PCI0_PLBBEARH	0xC 0EC80060	R	0x63 – 0x60	R	PCI PLB Slave Error Address Register High	458
PCI0_POM0LAL	0xC 0EC80068	R/W	0x6B – 0x68	R/W	PCI POM0 Local Address Low	459
PCI0_POM0LAH	0xC 0EC8006C	R/W	0x6F – 0x6C	R/W	PCI POM0 Local Address High	459
PCI0_POM0SA	0xC 0EC80070	R/W	0x73 – 0x70	R/W	PCI POM0 Size Attribute	459
PCI0_POM0PCIAL	0xC 0EC80074	R/W	0x77 – 0x74	R/W	PCI POM0 PCI Address Low	460
PCI0_POM0PCIAH	0xC 0EC80078	R/W	0x7B – 0x78	R/W	PCI POM0 PCI Address High	460
PCI0_POM1LAL	0xC 0EC8007C	R/W	0x7F – 0x7C	R/W	PCI POM1 Local Address Low	460
PCI0_POM1LAH	0xC 0EC80080	R/W	0x83 – 0x80	R/W	PCI POM1 Local Address High	461
PCI0_POM1SA	0xC 0EC80084	R/W	0x87 – 0x84	R/W	PCI POM1 Size Attribute	461
PCI0_POM1PCIAL	0xC 0EC80088	R/W	0x8B – 0x88	R/W	PCI POM1 PCI Address Low	461
PCI0_POM1PCIAH	0xC 0EC8008C	R/W	0x8F – 0x8C	R/W	PCI POM1 PCI Address High	462
PCI0_POM2SA	0xC 0EC80090	R/W	0x93 – 0x90	R/W	PCI POM2 Size/Attribute	462
PCI0_PIM0SAL	0xC 0EC80098	R/W	0x9B – 0x98	R/W	PCI PIM0 Size/Attribute Low	463
PCI0_PIM0SAH	0xC 0EC800F8	R/W	0xFB – 0xF8	R/W	PCI PIM0 Size/Attribute High	464
PCI0_PIM0LAL	0xC 0EC8009C	R/W	0x9F – 0x9C	R/W	PCI PIM0 Local Address Low	464
PCI0_PIM0LAH	0xC 0EC800A0	R/W	0xA3 – 0xA0	R/W	PCI PIM0 Local Address High	464
PCI0_PIM1SA	0xC 0EC800A4	R/W	0xA7 – 0xA4	R/W	PCI PIM1 Size/Attribute	465
PCI0_PIM1LAL	0xC 0EC800A8	R/W	0xAB – 0xA8	R/W	PCI PIM1 Local Address Low	465
PCI0_PIM1LAH	0xC 0EC800AC	R/W	0xAF – 0xAC	R/W	PCI PIM1 Local Address High	466
PCI0_PIM2SAL	0xC 0EC800B0	R/W	0xB3 – 0xB0	R/W	PCI PIM2 Size/Attribute Low	466
PCI0_PIM2SAH	0xC 0EC800FC	R/W	0xFF – 0xFC	R/W	PCI PIM2 Size/Attribute High	467
PCI0_PIM2LAL	0xC 0EC800B4	R/W	0xB7 – 0xB4	R/W	PCI PIM2 Local Address Low	467
PCI0_PIM2LAH	0xC 0EC800B8	R/W	0xBB – 0xB8	R/W	PCI PIM2 Local Address High	467
PCI0_OMCAPID	0xC 0EC800C0	R	0xC0	R	PCI Outbound MSI Capability Identifier	468
PCI0_OMNIPTR	0xC 0EC800C1	R/W	0xC1	R	PCI Outbound MSI Next Item Pointer	468
PCI0_OMMC	0xC 0EC800C2	R/W	0xC3 – 0xC2	R/W	PCI Outbound MSI Message Control	468
PCI0_OMMA	0xC 0EC800C4	R/W	0xC7 – 0xC4	R/W	PCI Outbound MSI Message Address	469
PCI0_OMMUA	0xC 0EC800C8	R/W	0xCB – 0xC8	R/W	PCI Outbound MSI Message Upper Address	469

Table 19-5. Internal PCI Configuration Registers (Continued)

Mnemonic	PLB		PCI Configuration		Register	Page
	Address	Access	Offset	Access		
PCI0_OMMDATA	0xC 0EC800CC	R/W	0xCD – 0xCC	R/W	PCI Outbound MSI Message Data	469
PCI0_OMMEOI	0xC 0EC800CE	R/W	0xCE	R/W	PCI Outbound MSI Message End Of Interrupt	470
PCI0_PMCAPID	0xC 0EC800D0	R/W	0xD0	R	PCI PMC Capability Identifier	470
PCI0_PMNIPTR	0xC 0EC800D1	R/W	0xD1	R	PCI PMC Next Item Pointer	470
PCI0_PMC	0xC 0EC800D2	R/W	0xD3 – 0xD2	R	PCI Power Management Capabilities	471
PCI0_PMCSR	0xC 0EC800D4	R/W	0xD5 – 0xD4	R/W	PCI Power Management Control Status	471
PCI0_PMCSRBSE	0xC 0EC800D6	R	0xD6	R	PCI PMCSR PCI to PCI Bridge Support Extensions	472
PCI0_PMDATA	0xC 0EC800D7	R	0xD7	R	PCI PMC Data Register	472
PCI0_PMSCRR	0xC 0EC800D8	R/W	0xD8	R/W	PCI Power Management State Change Request Register	472
PCI0_CAPID	0xC 0EC800DC	R	0xDC	R	PCI Capability Identifier	473
PCI0_NIPTR	0xC 0EC800DD	R	0xDD	R	PCI Next Item Pointer	473
PCI0_CMD	0xC 0EC800DE	R/W	0xDF – 0xDE	R/W	PCI Command	473
PCI0_STS	0xC 0EC800E0	R/W	0xE3 – 0xE0	R/W	PCI Status	474
PCI0_IDR	0xC 0EC800E4	R/W	0xE7 – 0xE4	R/W	PCI Internal Debug Register	474
PCI0_CID	0xC 0EC800E8	R	0xEB – 0xE8	R	PCI Internal Core Device ID	474
PCI0_RID	0xC 0EC800EC	R	0xEF – 0xEC	R	PCI Internal Core Revision ID	475

Registers listed in *Table 19-6* can be accessed in two ways:

1. By the processor using the PLB address
2. By an external PCI device using memory cycles to the proper address.

Table 19-6. PCI Simple Message Passing and Inbound MSI Registers

Mnemonic	PLB		PCI Memory		Register	Page
	Address	Access	Address	Access		
PCI0_MSGIL	0xC 0EC81100	R/W	(PCI0_BAR0H || PCI0_BAR0L) + (0x03 – 0x00)	R/W	PCI Message In Low	476
PCI0_MSGIH	0xC 0EC81104	R/W	(PCI0_BAR0H || PCI0_BAR0L) + (0x07 – 0x04)	R/W	PCI Message In High	476
PCI0_MSGOL	0xC 0EC81108	R/W	(PCI0_BAR0H || PCI0_BAR0L) + (0x0B – 0x08)	R/W	PCI Message Out Low	476
PCI0_MSGOH	0xC 0EC8110C	R/W	(PCI0_BAR0H || PCI0_BAR0L) + (0x0F – 0x0C)	R/W	PCI Message Out High	477
PCI0_IM	0xC 0EC811F8	W	(PCI0_BAR0H || PCI0_BAR0L) + (0xFB – 0xF8)	W	PCI Inbound MSI	477

19.14.4 PCI Standard Header Registers

The following registers (offset 0x00 – 0x3F) are part of the standard, PCI-defined, configuration header.

19.14.4.1 PCI Vendor ID Register (PCI0_VENDID)

PCI vendor ID register (PCI0_VENDID) is a 16-bit register used to identify the manufacturer of the PCI device. The local CPU (PLB master) can write a different value to this register.

Figure 19-10. PCI Vendor ID Register (PCI0_VENDID)

15:0	VID	Vendor ID	

19.14.4.2 PCI Device ID Register (PCI0_DEVID)

PCI Device ID register (PCI0_DEVID) is a 16-bit register used to identify the PCI device. The local CPU (PLB master) can write a different value to this register.

Figure 19-11. PCI Device ID Register (PCI0_DEVID)

15:0	DID	Device ID	

19.14.4.3 PCI Command Register (PCI0_CMD)

PCI command register (PCI0_CMD) is a 16-bit, read/write register used to control the operation of the PLB/PCI Bridge on the PCI bus.

Figure 19-12. PCI Command Register (PCI0_CMD)

15:11		Reserved	These bits are reserved, and return zeros when read.
10	INTD	Interrupt Disable	This bit disables the PLB/PCI Bridge from asserting INTA. A value of 0 enables the assertion of its INTA signal. A value of 1 disabled the assertion of its INTA signal. This bit is 0 at reset.
9	FBBC	Fast Back-to-Back Capable Read-only; returns 0 when read.	Fast back-to-back write enable. This bit enables PCI masters to perform fast back-to-back transactions to different devices. The PLB/PCI Bridge does not perform fast back-to-back transactions, therefore, this bit is read-only and returns zero when read.
8	SERE	SERR# Enable	Enable PCI0SErr. Enables driving PCI0SErr to report the detection of an error out to the PCI bus. If disabled, PCI0SErr is never asserted regardless of the detection of any error.
7	ADS	Address Stepping	Address stepping wait states. PLB/PCI bridge does not address step (except when generating a Configuration Type 0 cycle), therefore, this bit is read-only and returns zero when read.

6	PER	Parity Error Response	Parity error response. This bit enables the following types of PCI bus parity errors: PCI data bus parity errors while PCI is master PCI data bus parity errors while PCI is target PCI address bus parity errors When parity error response is disabled (set to 0), detection of these errors is masked and PERR is not asserted, although parity is still generated.
5	PLS	Palette Snooping	Enable special palette snooping. The PLB/PCI Bridge macro is not a VGA device, therefore, this bit is read-only and returns zero when read.
4	MWI	Memory Write and Invalidate	Enable memory write and invalidate command support. When high, PLB/PCI Bridge is allowed to generate MWI transactions. This bit is 0 at reset.
3	SPC	Special Cycle	Enable special cycle operations. PLB/PCI Bridge never monitors special cycles, therefore, this bit is read-only and returns zero when read.
2	PME	PCI Master Enable	Enables PLB/PCI Bridge to master cycles on the PCI bus. When this bit is 0, PLB-PCI Bridge only responds as a PLB slave to accesses to the internal register set.
1	MEMA	Memory Access	Controls response of PLB/PCI Bridge as a PCI memory target. A value of 1 enables PLB/PCI Bridge to respond as a target. This bit is 0 (disabled) at reset.
0	IOA	I/O Access	Controls response of PLB/PCI Bridge as a PCI I/O target. A value of 1 enables PLB/PCI Bridge to respond as a target. This bit is 0 (disabled) at reset.

19.14.4.4 PCI Status Register (PCI0_STATUS)

PCI status register (PCI0_STATUS) is a 16-bit, read/bit-reset register used to record status information for PCI bus related events. Bits in this register are only set as a result of specific events occurring on the PCI bus. They are reset by writing a 1 to the desired bit location. Writing a 0 to a bit location leaves that bit unchanged.

Figure 19-13. PCI Status Register (PCI0_STATUS)

15	DEPE	Detected Parity Error Write 1 to clear.	PLB/PCI Bridge sets this bit whenever it detects a PCI bus parity error. This bit is maskable only by bit 8 (PCI0_ERREN[PCIPO]) of the PCI Error Enable register. This bit is set when the following occurs: PCI address bus parity error detected when PLB/PCI Bridge is a target. PCI data bus parity error detected when a PCI master writes to PLB memory (PLB/PCI Bridge is the target). PCI data bus parity error detected when PLB/PCI Bridge masters a PCI read cycle. Writing a 1 to this bit resets it to 0.
14	SSE	Signaled System Error Write 1 to clear.	Signaled system error. PLB/PCI Bridge sets this bit if it asserts PCI0SErr. Writing a 1 to this bit resets it to 0.
13	RMA	Received Master Abort Write 1 to clear.	This bit is set whenever PLB/PCI Bridge terminates a PCI cycle for which it is the master with master abort (except configuration and special cycles) and the Master Abort Error Enable bit of the Error Enable Register is set (PCI0_ERREN[MAEE]=1). Writing a 1 to this bit resets it to 0.

12	RTA	Received Target Abort Write 1 to clear.	PLB/PCI Bridge sets this bit whenever a PCI cycle for which it is the master is terminated with target abort and the Target Abort Error Enable bit of the Error Enable Register is set (PCI0_ERREN[TAEE]). Writing a 1 to this bit resets it to 0.
11	STA	Signaled Target Abort Write 1 to clear.	PLB/PCI Bridge never signals a target abort. This bit is always 0.
10:9	DST	PCIDevSel Response Timing Read-only.	PCI_DEVSEL response timing. PLB/PCI Bridge asserts PCI_DevSel on the second clock (also known as medium response time) after PCI_Frame is asserted by a PCI master attempting to access memory on the PLB side of the bridge. These bits are read-only and always return 01b when read.
8	DPE	Data Parity Error Detected Write 1 to clear.	This bit is set when the following two conditions are met: PLB/PCI Bridge detects a data parity error (PCI0PErr asserted) when it is the master on a PCI read cycle, or it is the master when it samples PCI0PErr asserted on a PCI write cycle. The Parity Error Response bit is set (PCI0_CMD[PER]=1). Writing a 1 to this bit resets it to 0.
7	FBBC	Fast Back-to-Back Capable Read-only; returns one when read.	Indicates that the PCI target is capable of accepting fast back-to-back transactions when the transactions are not to the same agent. PLB/PCI Bridge target does accept this type of fast back-to-back transaction. This bit is read only and is set to one.
6		Reserved	This bit is hardcoded to zero. This bit was the UDF support bit in PCI 2.1.
5	66C	66 MHz Capable	Indicates that the device is able to run at 66 MHz. Hard coded to 1.
4	CL	Capabilities List	Indicates whether or not this device implements the pointer for a new capabilities linked list at offset 0x34. This bit is read-only and returns 1 when read, indicating that the value read at offset 34h is a pointer in configuration space to a linked list of new capabilities.
3	INTS	Interrupt Status	This read-only bit reflects the state of the outbound interrupt in the PLB/PCI Bridge. Only when the Interrupt Disable bit in the PCI0_CMD register is 0, and this Interrupt Status bit is 1, will the PLB/PCI Bridge's INTA signal be asserted. Setting the Interrupt Disable bit to 1 has no effect on the state of this bit.
2:0		Reserved	These bits are reserved, and return 0s when read.

19.14.4.5 PCI Revision ID Register (PCI0_REVID)

PCI revision ID register (PCI0_REVID) is an 8-bit register used to hold the current incremental revision number. The local CPU (PLB master) can write a value to this register.

Figure 19-14. PCI Revision ID Register (PCI0_REVID)

7:0	RID	Revision ID	Revision level of device.

19.14.4.6 PCI Class Register (PCI0_CLS)

PCI class register (PCI0_CLS) holds the class code. The local CPU (PLB master) can write a value to this register. (Class information is defined in the *PCI*, Version 2.3, Appendix D).

Figure 19-15. PCI Class Register (PCI0_CLS)

23:16	BASE	Base Class	Reset to 0x06, which indicates bridge device.

15:8	SUB	Subclass	Reset to 0x00, which indicates host bridge.
7:0	INT	Interface Class	Reset to 0x00.

19.14.4.7 PCI Cache Line Size Register (PCI0_CACHELS)

PCI cache line size register (PCI0_CACHELS) determines the size of a PCI cache line. PLB/PCI Bridge does not use this register for anything. MWI and MRL transactions issued by the PLB/PCI Bridge are hard coded to use a value of 64 bytes and 32 bytes respectively for the cache line size.

Figure 19-16. PCI Cache Line Size Register (PCI0_CACHELS)			
7:0	CLS	PCI Cache Line Size	

19.14.4.8 PCI Latency Timer Register (PCI0_LATTIM)

PCI latency timer register (PCI0_LATTIM) is an 8-bit read/write register used to hold the value of the PCI latency timer. The granularity of the latency timer is eight PCI cycles; therefore, the three low-order bits of this register are read-only and are hardwired to 0.

Figure 19-17. PCI Latency Timer Register (PCI0_LATTIM)			
7:0	LT	PCI Latency Timer	

19.14.4.9 PCI Header Type Register (PCI0_HDTYPE)

The PCI Header Type Register (PCI0_HDTYPE) (bits 0–6) is used to identify the second part of the PCI header (beginning at offset 0x10). It also determines whether a device contains multiple functions (bit 7). PLB/PCI Bridge implements the standard header and is not a multi-function device, therefore, PCI0_HDTYPE is read-only and returns 0x00 when read.

Figure 19-18. PCI Header Type Register (PCI0_HDTYPE)			
7:0	HT	PCI Header Type	

19.14.4.10 PCI Built-in Self Test (BIST) Control Register (PCI0_BIST)

PCI BIST register (PCI0_BIST) is used for control and status of BIST. PLB/PCI Bridge does not implement BIST, therefore, PCI0_BIST is read-only and returns 0x00 when read.

Figure 19-19. PCI BIST Control Register (PCI0_BIST)			
7:0	BIST	PCI Built-in-Self Test (BIST) Control	

19.14.4.11 PCI BAR0 Low Register (PCI0_BAR0L)

PCI BAR0 low register (PCI0_BAR0L) is enabled using the Enable bit of the PIM0 Size/Attribute Register. When disabled, PCI0_BAR0L is always zero (cannot be written). When enabled, PCI0_BAR0L defines the PCI characteristics of a region of PCI memory space that is mapped to PLB space. *Figure 19-20* describes PCI0_BAR0L register bits.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Note: The PIM0 Size/Attribute Low Register must be initialized by the local CPU before any PCI device is allowed to configure this register.

Figure 19-20. PCI BAR0 Low Register (PCI0_BAR0L)

31:12	BA	Base Address	These bits determine the lower 32 bits of the PCI Memory address space where this region is located. Only corresponding bits that are 1 in the size field of the PIM0 Size/Attribute Low Register are writable. Bits that are 0 in the size field of the PIM0 Size/Attribute Low Register cause the corresponding Base Address bits to be always 0.
11:4	BAZ	Base Address - always zero	These bits are always 0 since the minimum size of this range is 4KB.
3	PEF	Prefetchable	This bit determines if the region is prefetchable. Its value is determined by the PIM0 Size/Attribute Low, Prefetch Enable bit.
2:1	LT	Location Type	These bits are always 0b10 to indicate that the memory space can be located anywhere in the 64-bit address space.
0	MSI	Memory Space Indicator	This bit is always 0 to indicate memory space (rather than I/O).

19.14.4.12 PCI BAR0 High Register (PCI0_BAR0H)

The PCI BAR0 high register (P4CIX0_BAR0H) is enabled using the Enable bit of the PIM0 Size/Attribute register. When disabled, PCI0_BAR0H is always zero (cannot be written). When enabled, PCI0_BAR0H defines the PCI characteristics of a region of PCI memory space that is mapped to PLB space. This register defines the higher 32 bits of the PCI memory address space where this region is located. This register, along with the PCI BAR0 Low register, defines the 64-bit starting address of the PCI memory space that is mapped to the PLB space.

Note: The PIM0 Size/Attribute High Register must be initialized by the local CPU before any PCI device is allowed to configure this register.

Figure 19-21. PCI BAR0 High Register (PCI0_BAR0H)

31:0	BAH	Base Address High	Defines the upper 32 bits of PCI memory space that is mapped to PLB. Only corresponding bits that are 1 in the size field of the PIM0 Size/Attribute High Register are writable. Bits that are 0 in the size field of the PIM0 Size/Attribute High Register cause the corresponding Base Address bits to be always 0.

19.14.4.13 PCI BAR1 Register (PCI0_BAR1)

PCI BAR1 register (PCI0_BAR1) is enabled using the Enable bit of the PIM1 Size/Attribute Register. When disabled, PCI0_BAR1 is always zero (cannot be written). When enabled, PCI0_BAR1 defines the PCI characteristics of a region of PCI I/O space that is mapped to PLB space.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Note: The PIM1 Size/Attribute Register must be initialized by the local CPU before any PCI device is allowed to configure this register.

Figure 19-22. PCI BAR1 Register (PCI0_BAR1)

31:8	BA	Base Address	These bits determine the PCI I/O address space where this region is located.
7:2	BAZ	Base Address always zero	These bits are always 0x00 since the minimum size of this range is 256 bytes.
1		Reserved	Returns 0 when read.
0	MSI	Memory Space Indicator	This bit is always 1 to indicate I/O space (rather than memory).

19.14.4.14 PCI BAR2 Low Register (PCI0_BAR2L)

PCI BAR2 low register (PCI0_BAR2L) is enabled using the Enable bit of the PIM2 Size/Attribute Register. When disabled, PCI0_BAR2L is always zero (cannot be written). When enabled, PCI0_BAR2L defines the PCI characteristics of a region of PCI memory space that is mapped to PLB space. The Enable bit of the Expansion ROM BAR Register must be low for the PIM2 (BAR2) address space to be enabled.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Note: The PIM2 Size/Attribute Low Register must be initialized by the local CPU before any PCI device is allowed to configure this register.

Figure 19-23. PCI BAR2 Low Register (PCI0_BAR2L)

31:12	BA	Base Address	These bits determine the lower 32 bits of the PCI Memory address space where this region is located. Only corresponding bits that are 1 in the size field of the PIM2 Size/Attribute Low Register are writable. Bits that are 0 in the size field of the PIM2 Size/Attribute Low Register cause the corresponding Base Address bits to be always 0.
11:4	BAZ	Base Address - always zero	These bits are always 0x00 since the minimum size of this range is 4KB.
3	PEF	Prefetchable	This bit determines if the region is prefetchable. Its value is determine by the PIM2 Size/Attribute Low, Prefetch Enable bit.

2:1	LT	Location Type	These bits are always 0b10 to indicate that the memory space can be located anywhere in the 64-bit address space.
0	MSI	Memory Space Indicator	This bit is always 0 to indicate memory space (rather than I/O).

19.14.4.15 PCI BAR2 High Register (PCI0_BAR2H)

PCI BAR2 high register (PCI0_BAR2H) is enabled using the Enable bit of the PIM2 Size/Attribute Register. When disabled, PCI0_BAR2H is always zero (cannot be written). When enabled, PCI0_BAR2H defines the PCI characteristics of a region of PCI memory space that is mapped to PLB space. This register defines the higher 32 bits of the PCI memory address space where this region is located. This register, along with the PCI PIM2 BAR Low Register, defines the 64-bit starting address of the PCI memory space that is mapped to the PLB space.

Note: PCI0_BAR2H must be initialized by the local CPU before any PCI device is allowed to configure it.

Figure 19-24. PCI BAR2 High Register (PCI0_BAR2H)

31:0	BAH	Base Address High	Defines the upper 32 bits of PCI memory space that is mapped to PLB. Only corresponding bits that are 1 in the size field of the PIM2 Size/Attribute High Register are writable. Bits that are 0 in the size field of the PIM2 Size/Attribute High Register cause the corresponding Base Address bits to be always 0.

19.14.4.16 PCI Subsystem Vendor ID Register (PCI0_SBSYSVID)

PCI subsystem vendor ID register (PCI0_SBSYSVID)is a 16-bit register used to hold the Vendor ID for the subsystem or add-in board.

Note: PCI0_SBSYSVID holds the Vendor ID for PLB/PCI bridge that is used in many different designs.

Figure 19-25. PCI Subsystem Vendor ID Register (PCI0_SBSYSVID)

15:0	SVID	PCI Subsystem Vendor ID	

19.14.4.17 PCI Subsystem ID Register (PCI0_SBSYSID)

PCI Subsystem ID Register (PCI0_SBSYSID) is a 16-bit register used to hold the Device ID of the subsystem or add-in board.

Note: The Device ID Register holds the Device ID for PLB/PCI Bridge that can be used in many different designs.

Figure 19-26. PCI Subsystem ID Register (PCI0_SBSYSID)

15:0	SID	PCI Subsystem ID	

19.14.4.18 PCI Expansion ROM Base Address Register (PCI0_EROMBA)

PCI expansion ROM base address register (PCI0_EROMBA) is enabled using the Enable bit of the PIM2 Size/Attribute Low Register. When disabled, PCI0_EROMBA is always 0 (cannot be written). When enabled, this register is used as follows: If the Enable bit of PCI0_EROMBA is low, the PIM2 (BAR2) address space is enabled and the Expansion ROM address space is not. If the Enable bit of PCI0_EROMBA is high, the BAR2 address space is disabled, and the Expansion ROM address space is enabled, translated through PIM2. This register defines the PCI characteristics of a region of PCI memory space that is mapped to PLB space.

Note: The PIM2 Size/Attribute Low Register must be initialized by the local CPU before any PCI device is allowed to configure this register.

Figure 19-27. PCI Expansion ROM Base Address Register (PCI0_EROMBA)

31:12	BA	Base Address	These bits determine the PCI I/O address space where this region is located.
11:1	BAZ	Base Address - always zero	These bits are always 0 since the minimum size of this range is 4KB.
0	EN	Enable	When 1, the expansion ROM address space of the PLB/PCI Bridge is enabled and the BAR2 address space is disabled. When 0, the expansion ROM address space is disabled and the BAR2 address space is enabled, translated through PIM2.

19.14.4.19 PCI Capabilities Pointer (PCI0_CAP)

The PCI capabilities pointer register (PCI0_CAP) is a 8-bit pointer to the next capability in configuration space. This data structure is indicated in the PCI Status Register capabilities list bit (bit 4). This register points to the first item in the list of capabilities which points to the MSI capability structure at address offset 0xC0.

Figure 19-28. PCI Capabilities Pointer (PCI0_CAP)

7:0	CP	PCI Capabilities Pointer	

19.14.4.20 PCI Interrupt Line Register (PCI0_INTLN)

PCI Interrupt Line Register (PCI0_INTLN) is used to communicate interrupt line routing information. It controls nothing in the PLB/PCI Bridge.

Figure 19-29. PCI Interrupt Line Register (PCI0_INTLN)

7:0	IL	PCI Interrupt Line	

19.14.4.21 PCI Interrupt Pin Register (PCI0_INTPN)

PCI Interrupt Pin Register (PCI0_INTPN) tells which PCI interrupt line the device uses. This 8-bit register is read-only and the value 0x01 indicates INTA.

Figure 19-30. PCI Interrupt Pin Register (PCI0_INTPN)			
7:0	IP	PCI Interrupt Pin	

19.14.4.22 PCI Minimum Grant Register (PCI0_MINGNT)

PCI Minimum Grant register (PCI0_MINGNT) is used for specifying how long a burst period a PCI device needs. This 8-bit register defaults to 0x00. It is writable from the PLB side.

Figure 19-31. PCI Minimum Grant Register (PCI0_MINGNT)			
7:0	MG	PCI Minimum Grant	

19.14.4.23 PCI Maximum Latency Register (PCI0_MAXLTNCY)

PCI Maximum Latency Register (PCI0_MAXLTNCY) specifies how often a PCI device needs to gain access to the PCI bus. This 8-bit register defaults to 0x00 when read. It is writable from the PLB side.

Figure 19-32. PCI Maximum Latency Register (PCI0_MAXLTNCY)			
7:0	ML	PCI Maximum Latency	

19.14.5 Bridge Options Registers

The bridge options registers described under this section control miscellaneous functions of the PLB/PCI Bridge.

19.14.5.1 PCI Bridge Options 1 (PCI0_BRDGOPT1)

PCI Bridge Options 1 Register (PCI0_BRDGOPT1) controls various operating parameters of the PLB/PCI Bridge. These must be set up before transactions through the bridge are allowed in.

Note: Outbound read buffer assignment: The assignment of more than one master to a special purpose buffer can cause unpredictable results.

Figure 19-33. PCI Bridge Options 1 (PCI0_BRDGOPT1)			
31:29	RBP7	Outbound Read Buffer Mapping for PLB master 7:HSDMA 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
28:26	RBP6	Outbound Read Buffer Mapping for PLB master 6:DMA2P40 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
25:23	RBP5	Outbound Read Buffer Mapping for PLB master 5:PCI Express 0 Controller (PE0) 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
22:20	RBP4	Outbound Read Buffer Mapping for PLB master 4:Processor Data Cache Write Unit 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
19:17	RBP3	Outbound Read Buffer Mapping for PLB master 3:PCI Express 1 Controller (PE1) 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
16:14	RBP2	Outbound Read Buffer Mapping for PLB master 2: PCI Bridge Controller (PCI0) 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
13:11	RBP1	Outbound Read Buffer Mapping for PLB master 1: Processor Data Cache Read Unit 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
10:8	RBP0	Outbound Read Buffer Mapping for PLB master 0: Processor Instruction Cache Read Unit 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
7		Reserved	This bit must be set to 0.
6:5	PLRQ	PLB Request Priority 00 Lowest 01 Next highest 10 Next highest 11 Highest	This bit controls how the PLB/PCI Bridge PLB master controls the PLB arbitration priority for all PLB accesses.
4	MSEN	Inbound MSI Enable	This bit enables Inbound MSI. This is typically only used in Host-Bridge Mode. When high, the Inbound MSI logic drives the inbound interrupt to UIC.
3	PCAR	PCI Arbiter Park Mode	This bit defines how the internal PCI arbiter will handle bus parking. A value of 0 means that the arbiter will park on requester 0 (the bridge PCI master). A value of 1 means that the arbiter will park on the last master granted the bus. This bit must not be changed while PCI masters are active.
2:1	PTMR	PCI Target Memory Read command interpretation 00 Memory read 01 Memory read line 10 Memory read multiple 11 Reserved	This field allows PLB/PCI Bridge to be forced to treat a PCI memory read as a memory read multiple or memory read line with respect to the burst size implied by these read commands. This is for masters that use memory read for multiple beat bursts.
0		Reserved	Must be set to 0.

19.14.5.2 PCI Bridge Options 2 (PCI0_BRDGOPT2)

PCI Bridge Options 2 Register (PCI0_BRDGOPT2) controls various operating parameters of the PLB/PCI Bridge.

Note: Outbound read buffer assignment: The assignment of more than one master to a special purpose buffer can cause unpredictable results.

Figure 19-34. PCI Bridge Options 2 (PCI0_BRDGOPT2)

31:29	RBP11	Outbound Read Buffer Mapping for PLB master 11:PLB4 to AHB Bridge (PLB4XAHB) 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
28:26	RBP10	Outbound Read Buffer Mapping for PLB master 10:Media Access Layer (MAL) 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
25:23	RBP9	Outbound Read Buffer Mapping for PLB master 9:Security (EIP94) 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
22:20	RBP8	Outbound Read Buffer Mapping for PLB master 8:I2ODMA 000 256B general-purpose buffer #0 001 2KB special-purpose buffer #1	
19:17		Reserved	
16:15	FREQ	Frequency Range 00 Reserved 01 Reserved 10 33-66 MHz 11 0-33 MHz.	Read only value that reports the PCI frequency range
14		Reserved	Always read as 0.
13	EXWR	External Write to PCI Command Interrupt	When the PCI Command Register is written from the PCI side (inbound configuration write), this bit is set.
12:11		Reserved	Must be set to 0.
10	TDIS	PCI Discard Timer Disable	When 1, PLB/PCI Bridge never discards delayed read data.
9		Reserved	Must be set to 0.
8:7	INT1	PCI Interrupt Source 1	Reset to 0b01. Do not change.
6		Reserved	Must be set to 0.
5:4	INT0	PCI Interrupt Source 0	This field has a value of 00 after reset. Do not change. Resets to 00. Setting this field to 01 will mask the generation of outbound interrupts, both via INTA and MSI.
3	CPUW	Local CPU wait	This bit controls the value of the PCI local CPU wait (PCWE) strapping bit, which prevents the local CPU from running when this bit is 1. By causing this bit to be 1 at reset, the host PCI master is given time to initialize the internal register set of the PLB/PCI Bridge before the local CPU boots. This is typically used to set up boot code, when the local CPU is not booting from local ROM.

2	HTD	Host Configuration Enable Timer Disable	If this bit is set, the Host Configuration Enable timer never expires. See Host Configuration Enable bit below.
1	HTEX	Host Configuration Enable Timer Expired	This bit is set if the Host Configuration Enable timer expired. See Host Configuration Enable bit below.
0	HCEN	Host Configuration Enable	This bit controls host PCI access to the PCI configuration registers. If this bit is a 0, all host attempts to access PCI configuration registers (internal registers) of the PLB/PCI Bridge are retried. By causing this bit to be 0 at reset the local CPU is given time to initialize the internal register set before the host sees it. The reset value of this bit is determined by the PCI host configuration enable (PHCE) strapping bit. If PCI host configuration enable (PHCE) strapping bit is tied to a 1 (high), this bit has a value of 1 after reset. If PCI host configuration enable (PHCE) strapping bit is tied to a 0 (low), this bit has a value of 0 after reset. If this bit is cleared following reset, and the Host Configuration Enable Timer Disable bit is cleared, and if this bit does not get set by the local CPU within 2^{24} PLB clocks (126ms at 133MHz), then it is set automatically, and the Host Configuration Enable Timer Expired bit is set in this register. As per the PCI Specification, Version 2.3, as little as 500 ms is allowed for configuration to be set up. Thus, if PLB is run below 34MHz, this timer will not expire before this time period.

19.14.6 Error Handling Registers

The following registers are associated with error handling and reporting.

19.14.6.1 PCI Error Enable (PCI0_ERREN)

PCI Error Enable Register (PCI0_ERREN) is a 32-bit read/write register used to enable detection and reporting of errors for the PLB/PCI Bridge.

Figure 19-35. PCI Error Enable (PCI0_ERREN)

31:19		Reserved	Always read as zero.
18	REL	Route Error Locally	When set, errors are routed locally using ERROR_INT. Assertion of PLB MERR is not affected by this bit. When cleared, errors are routed to the PCI using PCI0SErr.
17	MRWE	MRdErr/MWrErr Assertion Enable	This bit enables the assertion of PLB Read MERR or PLB Write MERR when the PLB/PCI Bridge detects an error.
16	EAE	ERROR_INT Assertion Enable	This bit enables the assertion of ERROR_INT when the PLB/PCI Bridge detects an error.
15:13		Reserved	Set to zero.
12	RERE	MRdErr Receive Enable	This bit enables the detection of PLB Read MERR when the PLB/PCI Bridge is the PLB master. If this bit is set, the PLB/PCI Bridge may drive ERROR_INT to the local CPU or PCI0SErr to the PCI bus.
11	WERE	MWrErr Receive Enable	This bit enables the detection of PLB Write MERR when PLB/PCI Bridge is the PLB master. If this bit is set, the PLB/PCI Bridge may drive ERROR_INT to the local CPU or PCI0SErr to the PCI bus.
10	MQRE	MIRQ Receive Enable	This bit enables the detection of MIRQ when PLB/PCI Bridge is the PLB master. If this bit is set, the PLB/PCI Bridge may drive ERROR_INT to the local CPU or PCI0SErr to the PCI bus.
9	TRE	Timeout Receive Enable	This bit enables the detection of Timeout when PLB/PCI Bridge is the PLB master. If this bit is set, the PLB/PCI Bridge may drive ERROR_INT to the local CPU or PCI0SErr to the PCI bus.
8	PCIPO	PCI Parity Option	This bit should normally be left 1. It may be set to 0 to mask all types of PCI parity error checking. This bit is 1 after reset.
7	IDPE	PCI Inbound Write Data Parity Error Enable	This bit enables the detection of PCI data parity errors on inbound writes. **Note:** The Detected Parity Error bit of the PCI Status Register and the masking of PCI PERR is not affected by the above, but the assertion of ERROR_INT and PCI0SErr is.
6		Reserved	Always read as zero.
5	MAEE	Master Abort Error Enable	This bit enables the detection of master aborts as an error condition, when PLB/PCI Bridge is the PCI master. If this bit is set, PLB/PCI Bridge may drive PLB Read MERR or PLB Write MERR on the PLB or PCI0SErr on the PCI bus.
4	TAEE	Target Abort Error Enable	This bit enables the detection of a target abort received while PLB/PCI Bridge is the PCI master to be detected as an error condition. If this bit is set, the PLB/PCI Bridge may drive ERROR_INT to the local CPU or PCI0SErr on the PCI bus.
3		Reserved	Set to zero.
2	APEE	Address/Attribute Parity Error Enable	This bit enables the detection of Address or Attribute (PCI only) parity errors while the PLB/PCI Bridge is the PCI target. If this bit is set, the PLB/PCI Bridge may drive PCI0SErr on the PCI bus.

1	DRTE	Delayed Read Discard Timer Expired Error Enable	This bit enables the detection of Delayed Read Discard Timer expiration as an error. If this bit is set, the PLB/PCI Bridge may drive the ERROR_INT signal to the local CPU or PCI0SErr on the PCI bus.
0	SEE	PCI0SErr Received as Error Enable	This bit enables the detection of PCI0SErr as an error. If this bit is set, the PLB/PCI Bridge may drive the ERROR_INT signal to the local CPU. **Note:** The PLB/PCI Bridge may be receiving this error, while at the same time, it is driving PCI0SErr in response to some other error.

19.14.6.2 PCI Error Status (PCI0_ERRSTS)

PCI Error Status Register (PCI0_ERRSTS) is a 32-bit read/write register containing status on error conditions that have been detected. Bits in this register can only be set to 1 as a result of a system error occurring. These bits can be reset by writing a 1 to the desired bit location. Writing a 0 to any bit leaves that bit unchanged.

Figure 19-36. PCI Error Status (PCI0_ERRSTS)

31:18		Reserved	Always read as zero.
17	MRWE	MRdErr or MWrErr Signaled	This bit is set whenever an error occurs that causes PLB/PCI Bridge to assert PLB Read MERR or PLB Write MERR.
16	EIS	ERROR_INT Signaled	This bit is set whenever an error occurs that causes PLB/PCI Bridge to assert ERROR_INT.
15:13		Reserved	Always read as zero.
12	MRER	MRdErr Received	This bit is set when PLB/PCI Bridge is a PLB master and receives PLB Read MERR.
11	MWER	MWrErr Received	This bit is set when PLB/PCI Bridge is a PLB master and receives PLB Write MERR.
10	MQR	MIRQ Received	This bit is set when PLB/PCI Bridge is a PLB master and receives MIRQ.
9	TO	Timeout	This bit is set when PLB/PCI Bridge is a PLB master and receives Timeout.
8		Reserved	Always read as zero.
7	IPED	PCI Inbound Write Data Parity Error Detected	This bit is set when a PCI inbound write data parity error is detected and the PCI Inbound Write Data Parity Error Enable bit of the Error Enable Register is set.
6		Reserved	Always read as zero.
5		Reserved	Always read as zero. **Note:** The status bit for the detection of master aborts is in the PCI Status Register.
4		Reserved	Always read as zero. **Note:** The status bit for the detection of target aborts is in the PCI Status Register.
3		Reserved	Always read as zero.
2	APED	Address/Attribute Parity Error Detected	This bit is set when the PLB/PCI Bridge is the target on the PCI bus and detects an address or attribute parity error and the Address/Attribute Parity Error Enable bit of the Error Enable Register is set and the Parity Error Response bit of the PCI Command Register is set.
1	DTE	Delayed Read Discard Timer Expired	This bit is set when the PLB/PCI Bridge detects that the Discard Timer Expires bit and the Delayed Read Discard Timer Expired Error Enable bit of the Error Enable Register is set.

0	PSEE	$\overline{\text{PCI0SErr}}$ Received	This bit is set when $\overline{\text{PCI0SErr}}$ is asserted and the $\overline{\text{PCI0SErr}}$ Receive as Error Enable bit of the Error Enable Register is set.

19.14.6.3 PCI PLB Slave Error Attribute Register (PCI0_PLBBESR)

Figure 19-37. PCI PLB Slave Error Attribute Register (PCI0_PLBBESR)

31:10		Reserved	Always read as zero.
9	Val	Valid	Indicates if bits 8:0 are valid (This bit is only meaningful when an outbound error is indicated in the PCI0_ERRSTS register). **Note:** It is possible for a parity error to be detected too late for the PLB information to be saved.
8	RNW	Read, not write 0 Master supplies data to be written to the slave. 1 Slave supplies data to be read into the master.	RNW from the PLB master
7:4	PLBS	PLB Transfer Size 0000 Single beat transfer of one to four bytes of a 32-bit word starting at the target address 0010 Cache Line (32B) transfer 1100 Burst Transfer: n x 16 B/cycle where n = fixed length determined by master All other values Reserved or not supported.	Size from the PLB master
3:0	MID	MasterID	MasterID of the PLB master

19.14.6.4 PCI PLB Slave Error Address Low (PCI0_PLBEARL)

PCI PLB Slave Error Address Low Register (PCI0_PLBEARL) contains the lower 32 bits of address associated with an error on the PLB when PLB/PCI Bridge is a PLB slave. This register is read-only.

Figure 19-38. PCI PLB Slave Error Address Low (PCI0_PLBEARL)

31:0	EARL	PLB Slave Error Address Low	

19.14.6.5 PCI PLB Slave Error Address High (PCI0_PLBEARH)

PCI PLB Slave Error Address High Register (PCI0_PLBEARH) contains the higher 32 bits of address associated with an error on the PLB when PLB/PCI Bridge is a PLB slave. This register is read-only.

Figure 19-39. PCI PLB Slave Error Address High (PCI0_PLBEARH)

31:0	EARH	PLB Slave Error Address High	

19.14.7 POM Registers

The following registers control the mapping of PLB address space to PCI memory space.

19.14.7.1 PCI POM 0 Local Low Address (PCI0_POM0LAL)

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the POM registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-40. PCI POM 0 Local Low Address (PCI0_POM0LAL)

31:20	0LAL	Local Address Low	Defines the starting address of range 0 in PLB space that is mapped to PCI memory. Only the bits that are 1 in the POM 0 size are actually used to determine the starting address, all other bits are *don't cares*.
19:0		Reserved	Returns zero when read.

19.14.7.2 PCI POM 0 Local High Address (PCI0_POM0LAH)

Figure 19-41. PCI POM 0 Local High Address (PCI0_POM0LAH)

31:0	0LAH	Local Address High	Defines the upper 32 bits of the starting address of range 0 in PLB space that is mapped to PCI memory. All bits are writable.

19.14.7.3 PCI POM 0 Size/Attribute Register (PCI0_POM0SA)

PCI POM 0 size attribute register (PCI0_POM0SA) controls the size and attributes of the PLB space mapped to PCI memory for range 0.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the POM registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-42. PCI POM 0 Size/Attribute Register (PCI0_POM0SA)

31:20	SIZE	Size of POM0.	The size of the POM0 can range from 1MB to 4GB and must be a power of 2. To determine the value to program in this field, use the following process: 1. Represent the desired size with a 32-bit number. Use only one bit set in that number since it is a power of two. Use all 0s to represent 4GB. 2 Set all the bits to the left of the set bit. 3. Store bits 31:20 of the resulting number in this field. For example, if the desired POM0 size is 16M, start with 16x1024x1024 = 0x0100_0000. Set all the bits to the left of the set bit (0xFF00_0000). Bits 31:20 of that number are 0xFF0; therefore, 0xFF0 should be stored in this field.
19:1		Reserved	Always read as zero.

0	En	Enable Mapping 0 Disable 1 Enable	This bit determines if range 0 is enabled to map PLB space to PCI memory space. **Note:** The POM 0 Local Address, POM 0 PCI Low Address, and POM 0 PCI High Address must be initialized before enabling. This field has a value of 0 after reset.

19.14.7.4 PCI POM 0 PCI Address Low (PCI0_POM0PCIAL)

PCI0_POM0PCIAL along with PCI0_POM0CIAH defines the PCI address that is generated in response to PLB access to range 0. The register defines the lower 32 bits of the POM 0 PCI address. Only the bits that are 1 in the POM 0 Mask are actually passed to the PCI address. The other (least significant) bits of the PCI address are passed through from the PLB address.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the POM registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-43. PCI POM 0 PCI Address Low (PCI0_POM0PCIAL)

31:20	0PAL	PCI Low Address	
19:0		Reserved	Always read as zero.

19.14.7.5 PCI POM 0 PCI Address High (PCI0_POM0PCIAH)

PCI0_POM0PCIAH along with PCI0_POM0CIAL defines the PCI address that is generated in response to PLB access to range 0. The register defines the higher 32 bits of the POM 0 PCI Address.

Figure 19-44. PCI POM 0 PCI Address High (PCI0_POM0PCIAH)

31:0	0PAH	PCI High Address	

19.14.7.6 PCI POM 1 Local Address Low (PCI0_POM1LAL)

PCI0_POM1LAL defines the starting address of range 1 in PLB space that is mapped to PCI memory. See *PCI POM 0 Local Low Address (PCI0_POM0LAL)* on page 459 for details.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the POM registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-45. PCI POM 1 Local Address Low (PCI0_POM1LAL)

31:20	1LAL	Local Address Low	Defines the starting address of range 1 in PLB space that is mapped to PCI memory.
19:0		Reserved	Returns zero when read.

19.14.7.7 PCI POM 1 Local Address High (PCI0_POM1LAH)

PCI POM 1 local address high register (PCI0_POM1LAH) defines the upper 32 bits of the starting address of range 1 in PLB space that is mapped to PCI memory. See *PCI POM 0 Local High Address (PCI0_POM0LAH)* on page 459.

Figure 19-46. PCI POM 1 Local Address High (PCI0_POM1LAH)

31:0	1LAH	Local Address High	Defines the starting address of range 1 in PLB space that is mapped to PCI memory. Only the bits that are 1 in the POM 1 Size are actually used to determine the starting address. All other bits are don't cares.

19.14.7.8 PCI POM 1 Size/Attribute Register (PCI0_POM1SA)

PCI POM 1 size attribute register (PCI0_POM1SA) controls the size and attributes of the PLB space mapped to PCI memory for range 1.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the POM registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-47. PCI POM 1 Size/Attribute Register (PCI0_POM1SA)

31:20	SIZE	Size of POM 1	The size of the POM 1 can range from 1MB to 4GB and must be a power of 2. To determine the value to program in this field, use the following process: 1. Represent the desired size with a 32-bit number. Use only one bit set in that number since it is a power of two. Use all zeros to represent 4 GB. 2 Set all the bits to the left of the set bit. 3. Store bits 31:20 of the resulting number in this field. For example, if the desired POM 1 size is 16M, start with 16x1024x1024 = 0x0100_0000. Set all the bits to the left of the set bit (0xFF00_0000). Bits 31:20 of that number are FF0h; therefore, 0xFF0 should be stored in this field.
19:1		Reserved	Always read as zero.
0	En	Enable mapping 1 Enable 0 Disable	This bit determines if range 1 is enabled to map PLB space to PCI memory space. **Note:** The POM 1 Local Address, POM 1 PCI Low Address, and POM 1 PCI High Address must be initialized before enabling. This field has a value of 0 after reset.

19.14.7.9 PCI POM 1 PCI Address Low (PCI0_POM1PCIAL)

PCI POM 1 PCI address low register (PCI0_POM1PCIAL), along with PCI0_POM1PCIAH, defines the PCI address that is generated in response to PLB access to range 1. See section *PCI POM 0 PCI Address Low (PCI0_POM0PCIAL)* on page 460 for details.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the POM registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-48. PCI POM 1 PCI Address Low (PCI0_POM1PCIAL)			
31:20	1PAL	PCI Address Low	
19:0		Reserved	Returns zero when read.

19.14.7.10 PCI POM 1 PCI Address High (PCI0_POM1PCIAH)

PCI0_POM1PCIAH along with PCI0_POM1PCIAL defines the PCI address that is generated in response to PLB access to range 0. See *PCI POM 0 PCI Address High (PCI0_POM0PCIAH)* on page 460 for details.

Figure 19-49. PCI POM 1 PCI Address High (PCI0_POM1PCIAH)			
31:0	1PAH	PCI Address High	

19.14.7.11 PCI POM 2 Size/Attribute Register (PCI0_POM2SA)

PCI0_POM2SA described in *Figure 19-50* defines the attributes of the PLB space mapped to PCI memory for range 2.

Figure 19-50. PCI POM 2 Size/Attribute Register (PCI0_POM2SA)			
31:1		Reserved	Always read as zero.
0	En	Enables mapping 1 Enable 0 Disable	This bit determines if range 2 is enabled to map PLB space to PCI memory space. A value of 1 enables the mapping.

There are no POM 2 Local Address and POM 2 PCI registers. The values for these registers, translation map, and the size are hardcoded for POM 2. *Table 19-7* summarizes the hardcoded address map defined by POM 2.

Table 19-7. POM 2 Hardcoded Address Map

PLB Address Range	PCI Address Range
8000_0000_0000_0000 – FFFF_FFFF_FFFF_FFFF	0000_0000_0000_0000 Q – 7FFF_FFFF_FFFF_FFFF

19.14.8 PIM Registers

The following registers control how PCI memory space is mapped to PLB address space.

Note: The inbound transaction from the PCI to memory is defined in a window specified by the base address on the PCI, the PIM address translation to the PLB space, and the PIM size in the PCI0_PIMnSA register. For spaces defined by PIM0 and PIM2, the size must be lower than the PIM address except for address 0.

If the inbound transaction is performed outside of the defined window, no error will be reported for this operation.

19.14.8.1 PCI PIM0 Size/Attribute Register (PCI0_PIM0SAL)

PCI0_PIM0SAL defines the size and attributes of the region of PCI memory space that is mapped to local (PLB) space through PIM0. It affects the way PCI configuration register BAR0 works.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-51. PCI PIM 0 Size/Attribute Low Register (PCI0_PIM0SAL)

31:12	SIZL	Size Low of PIM0	The size of the PIM0 can range from 4KB to 2^{64} bytes and must be a power of 2. To determine the value to program in this field, use the following process: 1. Represent the desired size with a 64-bit number. Use only one bit set in that number since it is a power of two. Use all zeros to represent 2^{64} bytes. 2. Set all the bits to the left of the set bit. 3. Store bits [31:12] of the resulting number in this field. For example, if the desired PIM0 size is 16M, start with 16x1024x1024 = 0x0000_0000_0100_0000. Set all the bits to the left of the set bit (0xFFFF_FFFF_FF00_0000). Bits [31:12] of that number are 0xFF00_0; therefore, 0xFF00_0 should be stored in this field.
11:4		Reserved	Always read as zero.
3	MS	Map split using PIM1	This bit determines where the first 1KB of BAR0 is mapped when Split BAR0 Enable is active. If high, the first 1KB of BAR0 is mapped to the PLB, with the PLB address being determined by the PIM1 Local Address. This is typically used for I2O support, where an I2O function is located on the PLB. If low, the first 1 KB of BAR0 is mapped internally, enabling access to the Simple Message Passing and Inbound MSI. If Split BAR0 Enable is low, then this bit is a don't care.
2	SBE	Split BAR0 Enable	This bit enables splitting the BAR0-specified region into two ranges. When this bit is low, the entire BAR0 region is mapped to one region of PLB space, as determined by PIM0 Local Address registers. When this bit is high, the BAR0 range is divided into two regions: the first 1KB of the BAR0 range is mapped to either a PLB region as determined by the PIM1 Local Address registers, or to the internal registers for using Simple Message Passing and/or Inbound MSI. The second region is the remainder of the BAR0 range and is mapped to the PLB address space as determined by the PIM0 Local Address Register. If this bit is low, the entire BAR0 address space is mapped to PLB address space as determined by the PIM0 Local Address Register. This bit is 0 after reset.
1	PE	Prefetch Enable	This bit determines whether this region is prefetchable. The value of this bit is also readable in BAR0 low, bit 3.
0	En	Enable	If set, enables the PCI BAR0 registers and decoders.

19.14.8.2 PCI PIM0 Size/Attribute High Register (PCI0_PIM0SAH)

PCI0_PIM0SAH register defines the size and attributes of the region of PCI memory space that is mapped to local (PLB) space through PIM0. It affects the way PCI configuration register BAR0 works.

Figure 19-52. PCI PIM 0 Size/Attribute High Register (PCI0_PIM0SAH)

31:0	SIZ	Size High of PIM0	The size of the PIM0 can range from 4KB to 2^{64} bytes and must be a power of 2. To determine the value to program in this field, use the following process: 1. Represent the desired size with a 64-bit number. Use only one bit set in that number since it is a power of two. Use all zeros to represent 2^{64} bytes. 2. Set all the bits to the left of the set bit. 3. Store bits [63:32] of the resulting number in this field. For example, if the desired PIM0 size is 16M, start with 16x1024x1024 = 0x0000_0000_0100_0000. Set all the bits to the left of the set bit (0xFFFF_FFFF_FF00_0000). Bits 63:32 of that number are 0xFFFF_FFFF; therefore, 0xFFFF_FFFF should be stored in this field.

19.14.8.3 PCI PIM0 Local Low Address (PCI0_PIM0LAL)

PCI0_PIM0LAL defines the local (PLB) address that is generated in response to a PCI access to local (PLB) space through PIM0. Only the bits that are 1 in the size field of the PIM0 Size/Attribute Low Register are actually passed to the PLB address. The other (least significant) bits of the PLB address are passed through from the PCI address. Only bits 31:12 are writable, bits 11:0 are always zero.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-53. PCI PIM 0 Local Low Address (PCI0_PIM0LAL)

31:12	0LAL	Local Address Low	Defines the starting address of range 0 in PLB space that is mapped to PCI memory. Only the bits that are 1 in the PIM 0 Size/Attribute Low Register are actually passed to the PLB address.
11:0		Reserved	Returns zero when read.

19.14.8.4 PCI PIM0 Local High Address (PCI0_PIM0LAH)

PCI0_0LAH defines the upper 32 bits of the local (PLB) address that is generated in response to a PCI access to local (PLB) space through PIM0.

Only the bits that are 1 in the size field of the PIM0 Size/Attribute High Register are actually passed to the PLB address. The other (least significant) bits of the PLB address are passed through from the PCI address. All of these bits are passed to the upper PLB address.

Figure 19-54. PCI PIM 0 Local High Address (PCI0_PIM0LAH)			
31:0	0LAH	Local Address High	Defines the upper 32 bits of the starting address of range 0 in PLB space that is mapped to PCI memory. Only the bits that are 1 in the PIM 0 Size/Attribute High Register are actually passed to the PLB address.

19.14.8.5 PCI PIM1 Size/Attribute Register (PCI0_PIM1SA)

PCI0_PIM1SA defines the size and attributes of the region of PCI I/O space that is mapped to local (PLB) space through PIM 1. It affects the way PCI configuration register PIM1 BAR works.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-55. PCI PIM 1 Size/Attribute Register (PCI0_PIM1SA)			
31:8	SIZE	Size	SIZE defines the size of the region of PCI I/O space that is mapped to local (PLB) space through PIM 1. The size for the PIM1 is hardcoded to 256 bytes; therefore, read to these bits returns all ones. This size has no effect on the PIM0 1KB region.
7:1		Reserved	Returns zero when read.
0	EN	Enable	If set, enables the PCI BAR1 registers and decoder.

19.14.8.6 PCI PIM1 Local Low Address (PCI0_PIM1LAL)

PCI0_PIM1LAL defines the local (PLB) address that is generated in response to a PCI I/O access to local (PLB) space through PIM1. Only the bits that are 1 in size field of the PIM1 Size/Attribute Register are actually passed to the PLB address. The other (least significant) bits of the PLB address are passed through from the PCI address. Only bits 31:12 are writable, bits 11:0 are always 0. If the Split BAR0 Enable bit in the PIM 0 Size/Attribute Register has a value of 1, see *PCI PIM0 Size/Attribute Register (PCI0_PIM0SAL)* on page 463 for description of the register behavior regarding bit 2- Split BAR0 Enable.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-56. PCI PIM 1 Local Low Address (PCI0_PIM1LAL)			
31:12	1LAL	Local Address Low	Defines the starting address of range 1 in PLB space that is mapped to PCI I/O space. Only the bits that are 1 in the PIM 1 Size/Attribute Register are actually passed to the PLB address. Only bits 31:12 are writable.
11:0		Reserved	Returns zero when read.

19.14.8.7 PCI PIM1 Local High Address (PCI0_PIM1LAH)

PCI0_PIM1LAH defines the upper 32 bits of the local (PLB) address that is generated in response to a PCI I/O access to local (PLB) space through PIM1. The value of this register is passed on to the PLB upper address.

Figure 19-57. PCI PIM 1 Local High Address (PCI0_PIM1LAH)

31:0	1LAH	Local Address High	Defines the upper 32 bits of the starting address of range 1 in PLB space that is mapped to PCI I/O space. Only the bits that are 1 in the PIM 0 Size/Attribute High Register are actually passed to the PLB address.

19.14.8.8 PCI PIM2 Size/Attribute Low Register (PCI0_PIM2SAL)

PCI0_PIM2SAL defines the size and attributes of the region of PCI memory space that is mapped to PLB space through PIM2. It affects the way PIM2 BAR Register and PCI Expansion ROM BAR Register work.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512 MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-58. PCI PIM 2 Size/Attribute Low Register (PCI0_PIM2SAL)

31:12	SIZ	Size Low of PIM2	The size of the PIM2 can range from 4KB to 2^{64} bytes and must be a power of 2. To determine the value to program in this field, use the following process: 1. Represent the desired size with a 64-bit number. Use only one bit set in that number since it is a power of two. Use all zeros to represent 2^{64} bytes 2. Set all the bits to the left of the set bit. 3. Store bits 31:12 of the resulting number in this field. For example, if the desired PIM2 size is 16M, start with 16x1024x1024 = 0x0000_0000_0100_0000. Set all the bits to the left of the set bit (0xFFFF_FFFF_FF00_0000). Bits 31:12 of that number are 0xFF00_0; therefore,0xFF00_0 should be stored in this field.
11:2		Reserved	Returns zero when read.
1	PE	Prefetch Enable	This bit determines if this region is prefetchable. The value of this bit is also readable in BAR2 low, bit 3.
0	EN	Enable	If set, enables the PCI PIM2 BAR Register and the Expansion ROM BAR Register and the decoder.

19.14.8.9 PCI PIM2 Size/Attribute High Register (PCI0_PIM2SAH)

PCI0_PIM2SAH defines the size and attributes of the region of PCI memory space that is mapped to local (PLB) space through PIM2. It affects the way PCI configuration register BAR2 works.

Figure 19-59. PCI PIM 2 Size/Attribute High Register (PCI0_PIM2SAH)

31:0	SIZH	Size High of PIM2	The size of the PIM2 can range from 4KB to 2^{64} bytes and must be a power of 2. To determine the value to program in this field, use the following process: 1. Represent the desired size with a 64-bit number. Use only one bit set in that number since it is a power of two. Use all zeros to represent 2^{64} bytes. 2. Set all the bits to the left of the set bit. 3. Store bits 63:32 of the resulting number in this field. For example, if the desired PIM2 size is 16M, start with 16x1024x1024 = 0x0000_0000_0100_0000. Set all the bits to the left of the set bit (0xFFFF_FFFF_FF00_0000). Bits [63:32] of that number are 0xFFFF_FFFF; therefore, 0xFFFF_FFFF should be stored in this field.

19.14.8.10 PCI PIM2 Local Low Address (PCI0_PIM2LAL)

PCI0_PIM2LAL defines the local (PLB) address that is generated in response to a PCI access to local (PLB) space through PIM2. Only the bits that are 1 in the size field of the PIM2 Size/Attribute Low Register are actually passed to the PLB address. The other (least significant) bits of the PLB address are passed through from the PCI address. Only bits 31:12 are writable, bits 11:0 are always zero.

If the Enable bit (bit 0) in the PCI Expansion ROM BAR Register at 30h is set, then PIM2 uses the PCI Expansion ROM Base Address Register instead of the BAR2 Register.

PLB to PCI address mapping windows implemented in the PLB/PCI Bridge by the PIM/BAR registers are required to be naturally aligned in both address spaces. The legitimate values for the base addresses of the mapping windows are a function of the window size. For example, a 512 MB window has eight natural alignments at 0x0000 0000, 0x2000 0000, 0x4000 0000, 0x6000 0000, 0x8000 0000, 0xA000 0000, 0xC000 0000, and 0xE000 0000.

Figure 19-60. PCI PIM 2 Local Low Address (PCI0_PIM2LAL)

31:12	2LAL	Local Address Low	Defines the starting address of range 2 in PLB space that is mapped to PCI memory. Only the bits that are 1 in the PIM 2 Size/Attribute Low Register are actually passed to the PLB address. Only bits 31:12 are writable.
11:0		Reserved	Returns zero when read.

19.14.8.11 PCI PIM2 Local Address High (PCI0_PIM2LAH)

PCI0_PIM2LAH defines the upper 32 bits of the local (PLB) address that is generated in response to a PCI access to local (PLB) space through PIM2. Only the bits that are 1 in the size field of the PIM2 Size/Attribute High Register are actually passed to the PLB address. The other (least significant) bits of the PLB address are passed through from the PCI address.

If the Enable bit (bit 0) in the PCI Expansion ROM BAR Register at 30h is set, then PIM2 uses the PCI Expansion ROM Base Address Register instead of the BAR2 Register.

Figure 19-61. PCI PIM 2 Local Address High (PCI0_PIM2LAH)			
31:0	2LAH	Local Address High	Defines the upper 32 bits of the starting address of range 2 in PLB space that is mapped to PCI memory. Only the bits that are 1 in the PIM 2 Size/Attribute High Register are actually passed to the PLB address.

19.14.9 MSI Capability Block Definition Registers

This section describes the Outbound MSI (Message Signaled Interrupts) capability registers.

19.14.9.1 PCI MSI Capability Identifier Register (PCI0_OMCAPID)

PCI MSI Capability Identifier register (PCI0_OMCAPID) when read by system software as 05h indicates that the data structure currently being pointed to is the MSI capability structure.

Figure 19-62. PCI MSI Capabilities Identifier (PCI0_OMCAPID)		
7:0	CPID	Message Signaled Interrupts (MSI) Capabilities Identifier

19.14.9.2 PCI Next Item Pointer (PCI0_OMNIPTR)

PCI Next Item Pointer Register (PCI0_OMNIPTR) describes the location of the next item in the function's capability list. The Next Item Pointer default value points to the capability structure for PCI Power Management, located at D0h in the configuration space.

Figure 19-63. PCI MSI Next Item Pointer Register(PCI0_OMNIPTR)		
7:0	NPTR	Message Signaled Interrupts (MSI) Next Item Pointer

19.14.9.3 PCI Message Control Register (PCI0_OMMC)

PCI message control register (PCI0_OMMC) is a 16-bit register which provides system software control over MSI.

Figure 19-64. PCI Message Control Register (PCI0_OMMC)			
15:8		Reserved	Returns zero when read.
7	64AC	64-bit Address Capable	This bit is hardwired to a 1 and indicates that PLB/PCI Bridge is capable of generating a 64-bit message address.
6:4	MME	Multiple Message Enable	System software writes to this field to indicate the number of allocated messages for PLB/PCI Bridge. The number of allocated messages is aligned to a power of 2. Bit 6:5 of this field are hardwired to 0 and only bit 4 is writable. This field's state after reset is 0b000.

3:1	MMC	Multiple Message Capable	System software reads this field to determine the number of messages requested by PLB/PCI Bridge. The PLB/PCI Bridge requests two messages and is indicated by hardwiring this field to a value of 0b001.
0	MSI	MSI Enable 0 MSI disabled 1 MSI enabled	This read/write bit is used by the system to enable or disable MSI capability. This bit state after reset is 0 (MSI is disabled after reset).

19.14.9.4 PCI Message Address Register (PCI0_OMMA)

Figure 19-65. PCI Message Address Register (PCI0_OMMA)

31:2	MA	Message Address	Indicates system specific message address. If the Message Enable bit (bit 0 of the Message Control Register) is set, the contents of this register specify the doubleword aligned address pci_ad[31:2] for the MSI memory write transaction. pci_ad[1:0] are driven to zero during the address phase. This field is read/write.
1:0		Reserved	Returns zero when read.

19.14.9.5 PCI Message Upper Address Register (PCI0_OMMUA)

Figure 19-66. PCI Message Upper Address Register (PCI0_OMMUA)

31:0	MUA	Message Upper Address	PLB/PCI Bridge supports a 64-bit message address (bit 7 in Message Control Register is set to 1). The contents of this register specify the upper 32 bits of a 64-bit message address pci_ad[63:32]. If the contents of this register are zero, PLB/PCI Bridge uses the 32-bit address specified by the Message Address Register. This field is read/write and is configured by the system.

19.14.9.6 PCI Message Data Register (PCI0_OMMDATA)

Figure 19-67. PCI Message Data Register (PCI0_OMMDATA)

15:0	MD	Message Data	System-specified message. This field is written by the system. When PLB/PCI Bridge issues an MSI message, it writes this value to the previously defined message address. If PLB/PCI Bridge is allocated two messages by the system, the least significant bit of this field determines which message is being reported. If PLB/PCI Bridge is allocated only one message by the system, then PLB/PCI Bridge cannot modify this data on a MSI write.

19.14.9.7 PCI Message End of Interrupt Register (PCI0_OMMEOI)

Figure 19-68. PCI Message End of Interrupt Register (PCI0_OMMEOI)			
7:1		Reserved	Returns zero when read.
0	MEOI	Message End of Interrupt	Following the generation of an outbound MSI, the outbound MSI unit must be rearmed before another outbound MSI can be generated. Writing a 1 to this bit rearms the MSI function. This bit automatically clears itself (always reads zero). EOI is required only when using MSI.

19.14.10 Power Management Register Block Definition Registers

This section describes the PCI Power Management Interface registers.

19.14.10.1 PCI Power Management Capability Identifier Register (PCI0_PMCAPID)

PCI power management capability identifier register (PCI0_PMCAPID) when read by system software as 01h indicates that PLB/PCI Bridge supports power management and the data structure currently being pointed to is the PCI power management capability structure.*Figure 19-69* describes the PCI0_PMCAPID register bits.

Figure 19-69. PCI Power Management Capabilities Identifier (PCI0_PMCAPID)			
7:0	CPID	Capabilities Identifier	When read by system software as 0x01 indicates that PLB/PCI Bridge supports power management and the data structure currently being pointed to is the PCI power management capability structure.

19.14.10.2 PCI Power Management Next Item Pointer Register (PCI0_PMNIPTR)

PCI power management next item pointer register (PCI0_PMNIPTR) describes the location of the next item in the function's capability list. The Next Item Pointer defaults to pointing to the capability structure for PCI located at DCh in the configuration space.

Figure 19-70. PCI Power Management Next Item Pointer (PCI0_PMNIPTR)			
7:0	NIP	Next Item Pointer	Describes the location of the next item in the function's capability list. The Next Item Pointer defaults to pointing to the capability structure for PCI located at DCh in the configuration space. This can be overwritten with 00h if support for PCI is not desired.

19.14.10.3 PCI Power Management Capabilities Register (PCI0_PMC)

PCI power management capabilities register (PCI0_PMC) provides information on the capabilities of the function related to power management.

Figure 19-71. PCI Power Management Capabilities Register (PCI0_PMC)

15:11	PME	PME_Support	The PLB/PCI Bridge does not support PME#; therefore, all the bits are hardwired to 0.
10	D2	D2_Support	This bit determines if the D2 Power Management State is supported. PLB/PCI Bridge does not support the D2 Power Management State; therefore, this bit is hardwired to a 0.
9	D1	D1_Support	This bit determines if the D1 Power Management State is supported. PLB/PCI Bridge supports the D1 Power Management State; therefore, this bit is hardwired to 1.
8:6	AUX	Aux_Current	PLB/PCI Bridge does not support PME#; therefore, this field is unsupported. All the bits are hardwired to 0.
5	DSI	Device Specific Initialization (DSI)	The DSI bit indicates whether special initialization of this function is required (beyond the standard PCI configuration header) before the generic class device driver is able to use it. This bit is hardwired to 0.
4		Reserved	Returns zero when read.
3	PME	PME Clock	This bit is hardwired to 0 indicating that the function does not support PME# generation in any state.
2:0	V	Version	Returns a value of 010b on reads, indicating that this function complies with the *PCI Bus Power Management Interface Specification,* Version 1.1.

19.14.10.4 PCI Power Management Control/Status Register (PCI0_PMCSR)

PCI power management control/status register (PCI0_PMCSR) is a 16-bit register which is used to manage the PCI function's power management state as well as to enable or monitor PMEs. Host writes may be handled as delayed writes.

Figure 19-72. PCI Power Management Control/Status Register (PCI0_PMCSR)

15	PMES	PME_Status	The PLB/PCI Bridge does not support PME#; therefore, this bit is hard wired to a 0.
14:13	DSC	Data_Scale	The PLB/PCI Bridge does not support the Data Register; therefore, these bits are hard wired to 0b00.
12:9	DSE	Data_Select	The PLB/PCI Bridge does not support the Data Register; therefore, these bits are hard wired to 0b0000.
8	PME	PME_En	PLB/PCI Bridge does not support PME# generation; therefore, this bit is hard wired to a 0.
7:2		Reserved	Returns 0 when read.
1:0	PS	PowerState 00 D0 01 D1 10 D2 11 D3-Hot	This 2-bit R/W field is used both to determine the current power state of a function and to set the function into a new power state. If software attempts to write an unsupported, optional state to this field, the value of this field does not change. Writing this register may cause the PMSC_RR to change.

19.14.10.5 PCI PMCSR PCI-to-PLB/PCI Bridge Support Extensions (PCI0_PMCSRBSE)

PCI0_PMCSRBSE register is required for all PCI-to-PCI bridges. The PLB/PCI Bridge is not a PCI-to-PCI Bridge; therefore, it returns 0 when this register is read.

Figure 19-73. PCI PMCSR PCI-to-PLB/PCI Bridge Support Extensions (PCI0_PMCSRBSE)			
7:0	CSSE	Power Management Control/Status PCI-to-PLB/PCI Bridge Support Extensions.	Required for all PCI-to-PCI bridges. Always read as 0.

19.14.10.6 PCI Power Management Data (PCI0_PMDATA)

PCI power management data register (PCI0_PMDATA) is an optional register that provides a mechanism for the function to report state dependent operating data such as power consumed or heat dissipation. PLB/PCI Bridge does not implements this register; therefore, it returns 0 when this register is read.

Figure 19-74. PCI Power Management Data (PCI0_PMDATA)			
7:0	Zero	Optional register	Returns 0 when read.

19.14.10.7 PCI Power Management State Change Request Register (PCI0_PMSCRR)

PCI power management state change request register (PCI0_PMSCRR) provides a method of informing the local processor of a power management state change request and prevents the completion of the write to the PMCSR Register until the local processor indicates it is ready for the state change. This register is used with the registers in the capability structure for the power management, but is not a part of the power management capability structure.

Figure 19-75. PCI Power Management State Change Request Register (PCI0_PMSCRR)			
7:5		Reserved	Returns zero when read.
4	APW	Accept PMCSR Write	The local processor sets this bit when it is ready to change the power management state. The bit is cleared when the host configuration write to the PMCSR is accepted. This bit is always a 1 if the Delayed Write Enable bit (bit 0) is a 0. (It is possible for the local processor to write a 0 to this bit).
3	SCR	State Change Request	The PLB/PCI Bridge sets this bit when there is a host write to the PMCS for a power management state change request. This drives an interrupt to the local processor informing it of a state change request. The local processor must simultaneously clear this bit and set the Accept PMCSR Write bit when it is ready to change the state. After clearing this bit, new request will not be detected until the outstanding delayed write is accepted. (It is possible for the local processor to write a 1 to this bit.)
2:1	RS	Requested State	This field indicates the new power management state requested using the delayed host write to the PMCSR.
0	DWE	Delayed Write Enable 0 Immediate write 1 Delayed write	When 1, any configuration write to the PMCSR is completed as a delayed write. All writes to PMCSR are retried until the local processor sets the Accept PMCSR Write bit (bit 4). When 0, any configuration write to the PMCSR is completed immediately. This bit is a don't care if a host write to PMCSR requests a state change from D3hot to D0. The default state is 0 to prevent bus hang if the PM_Int service routine is not ready to go.

19.14.11 PCI Capability Block Definition Registers

This section describes the PCI Capability registers.

19.14.11.1 PCI Capability Identifier (PCI0_CAPID)

PCI capability identifier register (PCI0_CAPID) when read by system software as 0x07 indicates that the data structure currently being pointed to is the PCI capability structure.

Figure 19-76. PCI Capability Identifier (PCI0_CAPID)

7:0	CPID	Capability Identifier	

19.14.11.2 PCI Next Item Pointer Register (PCI0_NIPTR)

PCI Next Item Pointer Register (PCI0_NIPTR) describes the location of the next item in the capability list of the function. By default, it points to the VPD function at location 0xF0.This is the last item in the capability list; therefore, this field is hardwired to a value of 0x00.

Figure 19-77. PCI Next Item Pointer Register (PCI0_NIPTR)

7:0	NXIP	Next Item Pointer	

19.14.11.3 PCI Command Register (PCI0_CMD)

The reset value of bits 6:4 equal the reset value of the DMOS field of the PCI0_STS register, which in turn, depends on the number of special purpose outbound read buffers present.

Figure 19-78. PCI Command Register (PCI0_CMD)

15:4		Reserved	Returns zero when read.
3:2	MRBC	Maximum Memory-Read Byte Count	This field defaults to 00 to indicate that the PLB/PCI Bridge can request a maximum byte count of 512 bytes in a single memory read transaction. The system will not set this field higher than 00 because 512B is the designed maximum memory read byte count.
1	ERO	Enable Relaxed Ordering	This bit is hardwired to a 0, indicating that the PLB/PCI Bridge never sets the Relaxed Ordering attribute bit.
0	DPER	Data Parity Error Recovery Enable	This bit is writable but has no effect.

19.14.11.4 PCI Status Register (PCI0_STS)

Figure 19-79. PCI Status Register (PCI0_STS)			
31:29		Reserved	Returns 0 when read.
28:26	MCRS	Designed Max Cumulative Read Size Indicates the maximum cumulative outbound read byte count.	It varies dynamically, based on the values in the PCI command register bits 6:2.
25:23		Reserved	
22:21	MRBC	Designed Max Memory Read Byte Count	Hardwired to 0b10. The maximum read byte count generated by the PLB-PCI Bridge on outbound reads is 2048B.
20	DC	Device Complexity	This bit is hardwired to a 0 indicating that the PLB/PCI Bridge is a simple device. (It does not handle LOCK# and never retries.
19:16		Reserved	Returns 0 when read.
15:8	BN	Bus Number	This field indicates the number of the bus segment for the PLB/PCI Bridge containing this function. These bits are read/writable from the PLB side, but read only from the PCI side.
7:3	DN	Device Number	This field indicates the number of the device for the PLB/PCI Bridge containing this function. These bits are read/writable from the PLB side, but read only from the PCI side.
2:0	FN	Function Number	This field indicates the number of the function for the PLB/PCI Bridge containing this function. These bits are hardwired to 0b000.

19.14.11.5 PCI Internal Debug Register (PCI0_IDR)

This register is not used under normal operation. It is for debug purposes only. Its contents may be requested by support personnel. It is not accessible from the PCI bus using the configuration transaction. It is accessible from the PLB bus.

Figure 19-80. PCI Internal Debug Register (PCI0_IDR)			
31:0		Reserved	

19.14.11.6 PCI Internal Core Device ID Register (PCI0_CID)

This register is not used under normal operation. It is for debug purposes only. Its contents may be requested by support personnel. It is not accessible from the PCI bus using the configuration transaction. It is accessible from the PLB bus.

Figure 19-81. PCI Internal Core Device ID Register (PCI0_CID)			
31:0	CID	Internal Core Device ID	Read only.

19.14.11.7 PCI Internal Core Revision ID Register (PCI0_RID)

This register is not used under normal operation. It is for debug purposes only. Its contents may be requested by support personnel. It is not accessible from the PCI bus using the configuration transaction. It is accessible from the PLB bus.

Figure 19-82. PCI Internal Core Revision ID Register (PCI0_RID)			
31:0	RID	Internal Core Revision ID	Read only.

19.14.12 Vital Product Data (VPD) Registers

This describes the Vital Product Data (VPD) registers. Please refer to the *PCI Local Bus Specification, Version 2.3*, Appendix *I* for the detailed description and function of these registers.

19.14.12.1 PCI VPD Capability Identifier (PCI0_VPDCAPID)

PCI VPD capability identifier cap ID register (PCI0_VPDCAPID) when read by system software as 0x03, indicates that the data structure currently being pointed to is the VPD capability structure.

Figure 19-83. PCI VPD Capability Identifier (PCI0_VPDCAPID)			
7:0	VCID	Capability Identifier	

19.14.12.2 PCI VPD Next Item Pointer Register (PCI0_VPDNIPTR)

PCI VPD Next Item Pointer Register (PCI0_VPDNIPTR), with offset = 1, describes the location of the next item in the capability list of the function. This is the last item in the capability list; therefore, this field is defaults to a value of 0x00.

Figure 19-84. PCI VPD Next Item Pointer Register (PCI0_VPDNIPTR)			
7:0	NXIP	Next Item Pointer	

19.14.12.3 PCI VPD Address Register (PCI0_VPDADR)

Figure 19-85 describes the PCI0_VPDADR (offset = 2) register bits.

Figure 19-85. PCI VPD Address Register (PCI0_VPDADR)			
15	FBIT	Flag Bit	The flag bit is written from the PCI side when the VPD address is written. It is written by the PLB side to indicate the completion of a read or write operation. Please see the PCI Specification, Version 2.3, Appendix I for the detailed operation of the flag bit.
14:0	VADR	VPD Address	The VPD address is writable from the PCI side. It is used to specify the address of the VPD datum to be accessed. This field is not writable from the PLB side.

19.14.12.4 PCI VPD Data Register (PCI0_VPDDATA)

Figure 19-86 describes the PCI0_VPDDATA (offset = 4) register bits.

Figure 19-86. PCI VPD Data Register (PCI0_VPDDATA)			
31:0	VDAT	VPD Data	The VPD data register is writable from both the PCI and PLB sides. The PCI side puts data to be written to VPD in this register. The PLB side puts data that was read from VPD in this register.

19.14.13 Simple Message Passing and Inbound MSI Registers

The following registers are for the simple message passing mechanism and for inbound MSI.

Notes:

1. The registers are not accessible from the PCI side in configuration space, but rather in PCI memory space, using BAR0. They must be read or written with 32-bit transfers.
2. The registers are only accessible from the PCI side when the "Split BAR0 Enable" bit of the PIM0 Size/Attribute register is set.

19.14.13.1 PCI Message In Low (PCI0_MSGIL)

Figure 19-87. PCI Message In Low (PCI0_MSGIL)			
31:0	MIL	Message In Low	This holds an inbound message. If bit 0 is high, then an inbound interrupt is generated.

19.14.13.2 PCI Message In High (PCI0_MSGIH)

Figure 19-88. PCI Message In High (PCI0_MSGIH)			
31:0	MIH	Message In High	This holds the upper 32 bits of a 64-bit inbound message.

19.14.13.3 PCI Message Out Low (PCI0_MSGOL)

Figure 19-89. PCI Message Out Low (PCI0_MSGOL)			
31:0	MOL	Message Out Low	This holds an outbound message. If bit 0 is high, then an outbound interrupt is generated.

19.14.13.4 PCI Message Out High (PCI0_MSGOH)

Figure 19-90. PCI Message Out High (PCI0_MSGOH)			
31:0	MOH	Message Out High	This holds the upper 32 bits of a 64-bit outbound message.

19.14.13.5 PCI Inbound Message MSI (PCI0_IM)

PCI inbound message MSI (PCI0_IM) is enabled by setting the Inbound MSI Enable bit of the Bridge Options 1 register. Address mapping must be also be setup appropriately.

Figure 19-91. PCI Inbound Message MSI (PCI0_IM)			
31:4		Reserved	Returns zero when read.
3:0	IMSI	MSI In	Inbound MSIs write this register. The data value written causes the corresponding pci_msi_int0:12 to go active to the interrupt controller.

19.15 Boot Modes

There are three supported methods of booting the local (PLB) CPU. The method chosen depends upon where the boot code is located and how the boot code is initialized. The methods are:

- Boot from local ROM, when in host-bridge mode
- Boot from local ROM, when in adapter-bridge mode
- Boot from PCI, when in adapter-bridge mode (PPC460EX and PPC460GT only)

The boot sequence of most PCI systems includes a few steps that are relevant to this Boot Modes discussion. This first such step is the host CPU initializing its PCI host bridge. In the next step, the host CPU runs the PCI configuration sequence, during which all the adapter's configuration registers are read and initialized. This step requires many of the configuration registers to contain values that indicate things about the adapter. The final step is when the host CPU boots its operating system.

The register set of the PLB/PCI Bridge includes the following registers:

- Vendor ID
- Device ID
- Revision ID
- Class
- Subsystem
- Vendor ID
- Subsystem Vendor ID
- Subsystem Device ID

The above register are read by the PCI configuration sequence to gather information about the adapter when in adapter-bridge mode. The register set of the PLB/PCI Bridge also includes the BAR0, BAR1, BAR2 and Expansion ROM BAR registers that behave in a manner that indicates to the PCI configuration sequence memory space needs of the adapter. The behavior of these registers is governed by the values of the PIM0, PIM1, and PIM2 Size/Attribute registers. These registers are referred to as "adapter's needs" registers.

The values of these "adapter's needs" registers can be written from either the local CPU (PLB) or the host (PCI bus).

19.15.1 Booting From Local ROM, When in Host-Bridge Mode

When the PLB/PCI Bridge is used as the host-bridge, the local CPU must boot from local ROM (the local CPU's access to the local ROM does not involve the PLB/PCI Bridge). As part of the boot process, the local CPU initializes the registers of the PLB/PCI Bridge before PCI activity begins. Generally, the configuration registers of the PLB/PCI Bridge are only accessed by the local CPU, never from the PCI side.

In this mode, the "adapter's needs" registers are not read (since the PLB/PCI Bridge is not an adapter). Instead, the boot code contains all necessary information needed for initializing the PLB/PCI Bridge. Thus, it is not necessary to have any specific values strapped into the registers.

Once the registers of the PLB/PCI Bridge are initialized, PCI activity may begin. At some point after this, the local CPU runs the PCI configuration sequence during which it determines what devices are attached to the PCI bus and initializes their configuration registers.

19.15.2 Booting From Local ROM, When in Adapter-Bridge Mode

When the PLB/PCI Bridge is used in adapter-bridge mode, the most common method of booting is booting from local ROM or flash (the local CPU's access to the local ROM does not involve the PLB/PCI Bridge). Typically, this boot process goes on in parallel with the host booting.

As part of the host's boot process, it typically runs the PCI configuration sequence. However, before the PCI configuration sequence can run, the configuration registers of the PLB/PCI Bridge must be set to indicate the "adapter's needs." This can be done by having the local CPU write to the registers.

If the values are set using local CPU writes, then it must be ensured that the host does not proceed to the PCI configuration sequence before the local CPU completes its initialization. This is guaranteed by strapping the PHCE strapping bit low, causing the Host Configuration Enable bit of the Bridge Options 2 Register to be cleared at reset. When cleared, host accesses to the configuration registers are retried. This ensures that the local CPU has time to initialize the "adapter's needs" registers. When initialization is complete, the local CPU writes the "Host Configuration Enable" bit high, allowing the host CPU to complete the PCI configuration sequence.

19.15.3 Booting From PCI, When in Adapter-Bridge Mode

When the PLB/PCI Bridge is used in adapter-bridge mode, it can be enabled to allow the local CPU to boot from memory on the PCI bus. The typical boot sequence for this mode is as follows:

1. The local CPU is initially prevented from booting while the host boots. This is done by strapping the PCWE strapping bit high, which causes the "Local CPU Wait" bit of the Bridge Options 2 Register to be high at reset.
2. The host CPU boots and runs the PCI initialization sequence. Thus, the "adapter's needs" registers of the PLB/PCI Bridge must be strapped appropriately.
3. Eventually (probably after the operating system boots), the host runs a program that puts boot code for the adapter in PCI memory space and initializes the POM0 registers of the PLB/PCI Bridge to point to the boot code.

4. The host then enables the local CPU to boot. This is done by writing a 0 to the "Local CPU Wait" bit of the Bridge Options 2 Register.

Notes:

1. If the PLB contains another interface controller that can be a boot device (such as a flash controller), then it must be disabled.
2. Booting from the PCI immediately after reset (ROM on PCI bus) is not supported.
3. To ensure that there are no deadlocks, the PLB arbiter must be set to FAIR, and the PLB priority of all PLB masters that access the PLB/PCI Bridge must be set to the same PLB priority as the PLB/PCI Bridge.

19.15.4 Option ROM Support

The PLB/PCI Bridge supports PCI option ROM by allowing a region of PCI memory space to be mapped to local (PLB) memory (SDRAM or ROM). The region of PCI memory space is determined by the "Expansion ROM BAR" at offset 30h in PCI configuration space. The PLB/PCI Bridge allows the use of either PIM2 or Option ROM, not both. The "Expansion ROM Enable" bit in the PCI Expansion ROM Base Address register determines the choice. See *PCI Expansion ROM Base Address Register (PCI0_EROMBA)* on page 451.

19.16 Software Initialization

The PLB/PCI Bridge has a set of registers that must be initialized before general operation begins.

19.16.1 Address Map Initialization

When the PLB/PCI Bridge is the master on the PCI bus, it can generate memory, I/O, configuration and special cycles. The PLB address ranges used to generate these cycle are all fixed, except for memory cycles. PCI memory cycles range sizes and PLB address space is specified in the POM registers. Also, the address of the resulting PCI memory cycle may be an offset from the PLB address (address translation occurs). This translation is also specified in the POM registers. The POM registers do not default to usable values following reset; they must be initialized before attempting to generate PCI memory cycles.

When the PLB/PCI Bridge is a target on the PCI bus, it can respond to memory cycles. The memory cycle address ranges that the PLB/PCI Bridge responds to are specified in the BAR0, BAR1 and BAR2 registers. The BAR registers are typically initialized as part of the standard PCI initialization process; however, before the BAR registers are initialized, the PIM Size/Attribute registers must be set to control the BAR behavior. The PIM Size/Attribute registers may be strapped or written to specify a usable value.

Before inbound traffic begins, the PIM Local Address registers must be specified to control the resulting PLB address (translated address). The PIM Local Address registers do not default to usable values following reset; they must be initialized before responding as a PCI memory target is allowed.

19.16.2 Other Registers that must be Initialized

- Error handling is initially disabled (error detection masked). If error handling is to be enabled, then the Error Enable Register must be initialized appropriately.
- The Bridge Options 1 Register and the Bridge Options 2 Register contain a variety of options that may need to be changed.

19.16.3 Adapter-Bridge Mode Configuration Access

When the PLB/PCI Bridge is used as a PCI adapter interface (in adapter-bridge mode), it can respond as a configuration target. The PCI_IDSEL pin must be hooked up (typically only done in adapter-bridge mode) and active for the PLB/PCI Bridge to respond as a configuration target. Before the host runs the PCI Configuration sequence, the following registers must have the appropriate values.

- PCI Vendor ID
- PCI Device ID
- PCI Revision ID
- PCI Class
- PCI Subsystem ID
- PCI Subsystem Vendor ID

They can be written by the local CPU with values taken from a local ROM. If these values are written by the local CPU, the Host Configuration Enable bit in the Bridge Options 2 register must be 0 to prevent the host CPU from reading these values before the local CPU has written them.

19.17 PCI–PLB Relative Clock Frequency Requirements

The relative frequency requirements between PLB clock and PCI clock are:

0 ≤ PCI clock frequency and 1.5 x PCI clock frequency ≤ PLB clock frequency

For example, if PCI clock is 66MHz, then the PLB clock must be 100MHz or greater.

20. PCI Express (PCIE)

This section describes the operation of PCI Express Revision 1.1 in the PPC460EX/EXr/GT.

Features include:

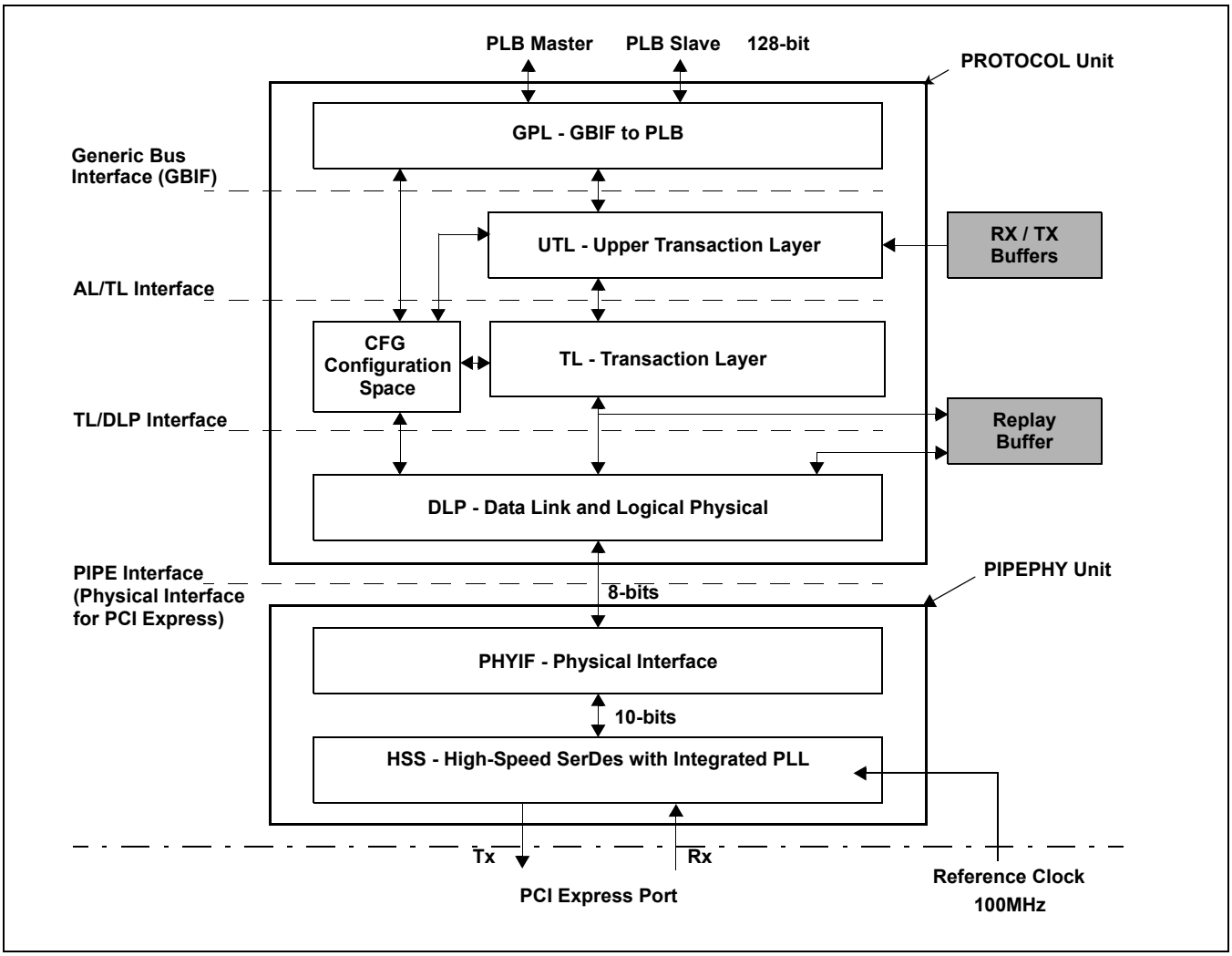
- Two independent PCI Express interfaces
  - One 4 lanes
  - One 1 lane
  - 2.5 Gbps full duplex per lane
- Compliant with PCI Express base specification 1.1
- Each PCI Express port can be End Point or Root Complex. (Upstream & Downstream)
  - Applications compliant with MSI rules are limited to one Endpoint port per PPC460EX/EXr/GT
- Power Management
- Supports one virtual channel (VC0) no Traffic Class (TC) filtering
- Maximum Payload block size 512 Bytes
- Supports up to 512 Bytes maximum Read request size
- Requests supported:
  - Up to 4 (x4) or 2 (x1) posted outbound Write requests (memory and messages)
  - Up to 4 (x4) or 2 (x1) posted inbound Write requests
  - Up to 4 (x4) or 2 (x1) outbound Read requests outstanding on PCI Express
  - Up to 4 (x4) or 2 (x1) inbound Read requests outstanding on PCI Express
  - Outbound I/O request as a PCI Express Root Port
  - Inbound I/O request as a PCI Express Endpoint
- Buffering in each PCI Express port for the following transaction types:
  - 2KB Replay buffer: up to 4 in flight transactions
  - 2KB (x4) or 1KB (x1) for Outbound posted Writes
  - 2KB (x4) or 1KB (x1) for Outbound Reads completion
  - 2KB (x4) or 1KB (x1) for Inbound posted Writes
  - 2KB (x4) or 1KB (x1) for Inbound Reads completion
- Parity checking on each buffer
- Programmable Outbound Memory (POM) regions: 3 memory, 1 I/O, 1 message, 1 configuration, 1 internal registers
- Programmable Inbound Memory (PIM) regions: 4 memory, 1 I/O, 1 expansion ROM
- INTx Interrupts support (legacy PCI):
  - Up to four INTx Termination for Root Ports. A/B/C/D interrupts are wired to the UIC
  - A/B/C/D INTx types generation for Endpoints
- MSI - Message Signaled Interrupts
  - MSI generation for Endpoint
  - MSI termination for Root Ports
  - MSI_X termination for Root Ports

20.1 Overview

The PPC460EX/EXr/GT provides two independent PCI Express ports, which are interconnected by means of the on-chip PLB bus. One interface can be configured as one to four lanes while the other functions as one-lane only. Both can be Root or Endpoint Ports. The single lane interface shares a High-Speed SerDes with the Serial ATA (SATA) interface.

The on-chip connection enables communication with other devices on the PLB, for example, the SDRAM memory controller and the PPC440 processor. This allows an inbound memory transaction on one of the PCI Express ports to progress along the PLB and appear as if it were an outbound request on one of the other PCI Express ports. The logic is similar to a PCI opaque bridge. *Figure 20-1* shows the PCI Express stack implementation in the PPC460EX/EXr/GT.

Figure 20-1. PCI Express Stack Implementation



20.2 Compliance with the PCI Express Specification

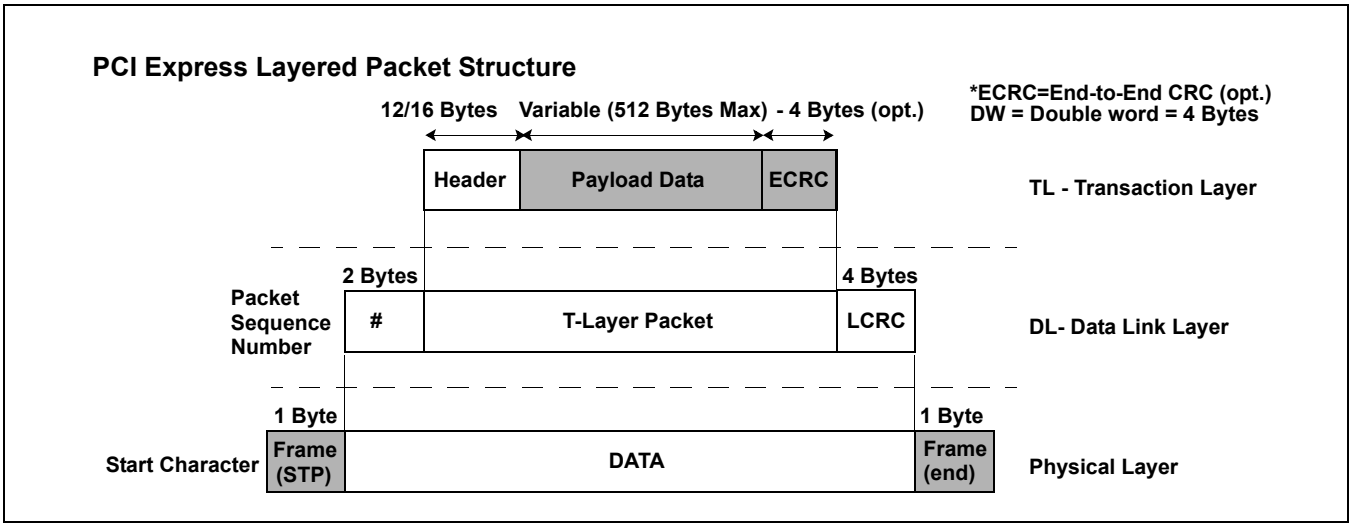
The PPC460EX/EXr/GT implementation of PCI Express complies with the *PCI Express Base Specification*, Revision 1.1 (March 8, 2005).

This document defines the terms *outbound* and *inbound* as describing the direction of traffic through the PLB-PCI Bridge. The PLB bus is considered *in* and the PCI bus is considered *out*. Thus, an outbound transaction is a read or write in which the PLB-PCI Bridge is the slave on the PLB bus and the initiator on the PCI bus. An inbound transaction is a read or write in which the PLB-PCI Bridge is the target on the PCI bus and the master on the PLB bus.

20.3 PCI Express Packet-Based Protocol

PCI Express is a packet-based protocol, which promotes data integrity during link transmission, as illustrated in *Figure 20-2*.

Figure 20-2. PCI Express Layered Packet Structure



20.4 PCI Express Stack Features

The following list describes the PCI Express Stack Features.

- Each PCI Express port can be an Endpoint or Root Complex (Upstream and Downstream). Because of the Message Signaled Interrupt (MSI) limitation, only one Endpoint port per PPC460EX/EXr/GT is supported.
- Single function device support per PCI express port when configured as an Endpoint (one Endpoint configuration space per stack)
- Each port is programmable as either a Root Port or Endpoint
- Supports end-to-end data integrity by means of end-to-end cyclic redundancy check (ECRC) generation and checking
- Supports PCI Express Advanced Error Reporting
- Supports one virtual channel (VC0) with no Traffic Class (TC) filtering
- PCI Express message support and routing
- Supports up to 512-byte maximum payload size in PCI Express transactions
- Supports up to 512-byte maximum read request size
- Requests:
  - Supports up to two (x1) or four (x4) posted outbound write requests (memory and messages)
  - Supports up to two (x1) or four (x4) posted inbound write requests
  - Supports up to two (x1) or four (x4) outbound read requests outstanding on PCI Express
  - Supports up to two (x1) or four (x4) inbound read requests outstanding on PCI Express
  - Supports outbound I/O request as a PCI Express Root Port
  - Supports inbound I/O request as a PCI Express Endpoint
- All PCI Express stacks provide buffering for the following transaction types:
  - 2-KB replay buffer supporting up to eight in-flight transactions
  - 1 KB (x1) or 2 KB (x4) for outbound posted writes
  - 1 KB (x1) or 2 KB (x4) total special purpose buffer for outbound read completion with:
    - Access controlled by four buffer handlers
    - Up to four bus masters assigned to the buffer handlers
  - 256-byte outbound read general purpose buffer

  - 1 KB (x1) or 2 KB (x4) for inbound posted writes
  - 1 KB (x1) or 2 KB (x4) for inbound reads completion
- INTx interrupt support (PCI legacy):
  - Supports termination of up to four INTx types for Root Ports. A/B/C/D interrupts are wired to the UIC.
  - Supports generation of A/B/C/D INTx types for Endpoints
- MSI - Message Signaled Interrupts
  - Supports MSI Generation for Endpoint
  - Supports MSI Termination for Root Ports
  - Supports MSI_X Termination for Root Ports
- Link-specific features
  - Automatic receiver polarity reversal
  - Link training control features supported as a Root Port
  - Generation of link reset, link disable, loopback, and scrambling disable
  - Reception of loopback and scrambling disable
  - Link training control features supported as an Endpoint port
  - Generation of loopback and scrambling disable
  - Reception of link reset, disable, loopback, and scrambling disable
  - Link failure error recovery
- Address Translation
  - POM Programmable Outbound Memory Regions: 3 memory, 1 I/O, 1 message, 1 configuration, 1 internal
  - PIM Programmable Inbound Memory Regions: 4 memory, 1 I/O, 1 expansion ROM

20.5 PCI Express Features Not Supported in the PPC460EX/EXr/GT Implementation

- Port bifurcation
- Booting PPC440 processor over the PCI Express bus after the host has booted
- Multiple virtual channels
- Traffic class filtering
- Configuration space
- Virtual channel extended capability structure
- Device serial number extended capability structure
- Power budgeting extended capability structure
- Line read from CPU. Only single beat read is supported from CPU.
- PCI Express switch/transparent bridge capability between PCI ports
- MSI-X generation
- Generate configuration cycle as an Endpoint

20.6 PCI Express Function

The PCI Express stack has been designed with the following functional components:

- Physical Unit including HSS (High Speed Serial Link) and Intermediate frequency PLL
- DPL Data Link and Logical Physical
- TL Transaction Layer
  - Configuration Registers
- UTL Upper Transaction Layer

- PLB Master and Slave Interface

20.6.1 Little Endian and Big Endian Ordering

The PCI Express function uses little endian byte ordering on the PCI Express link, and big endian byte ordering on the PLB bus. This results in big endian byte ordering for data located within the memory space or the PPC440 processor ordered in big endian.

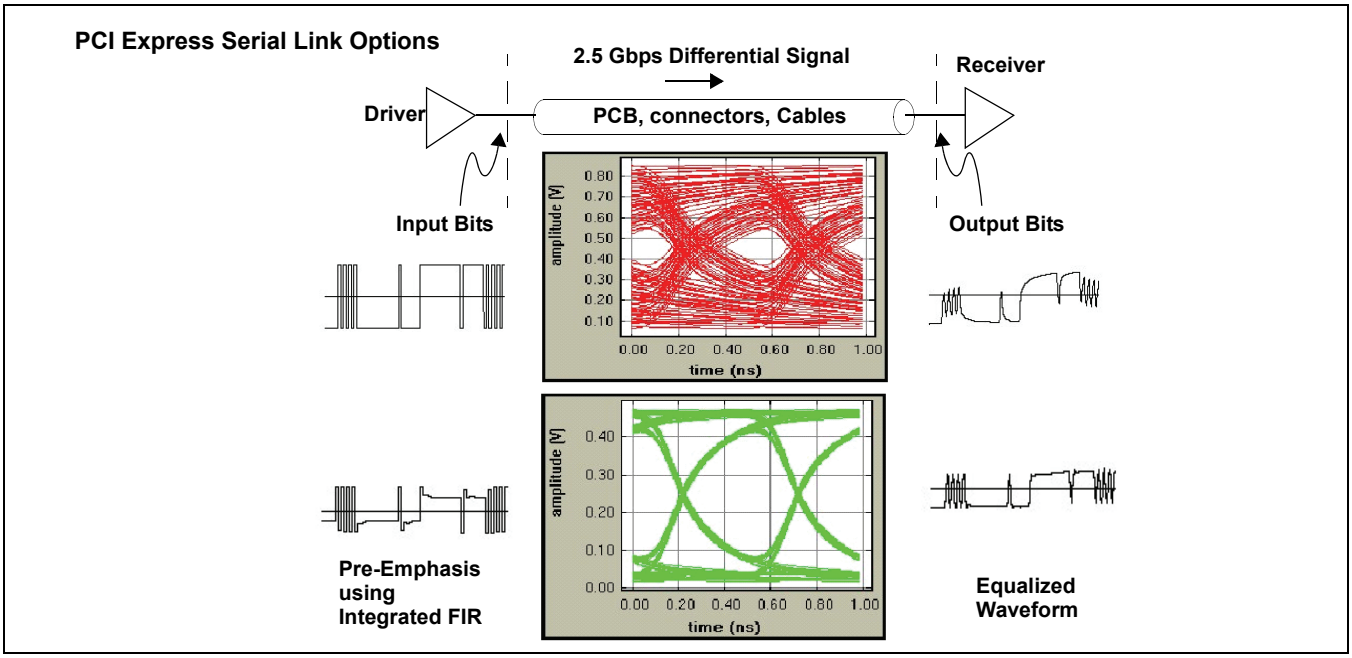
Byte swapping is performed in the PCIEXUTL function to maintain compatibility between big endian and little endian spaces.

20.7 High Speed Serial Link

The High Speed SerDes (HSS) for PCI Express consists of 2.5Gbps serial link function which contains:

- A common block that generates all required bias currents, contains startup and auto calibration logic, and contains the PLL which multiplies the reference clock to the required Serial Link Rate(s) and provides the reference clock for each of the Clock/Data Recovery (CDR) units.
- A transmit block that consists of a serializer, a differential CML driver with selectable PCI Express de-emphasis and swing levels, and a BIST pattern generator.
- A receive block that consists of a differential CML receiver with equalization, clock and data recovery circuitry, signal detection, deserializer and BIST pattern verifier.

Figure 20-3. PCI Express High Speed Serial Link



To improve transmission quality, the following enhancements have been made at the driver and receiver levels:

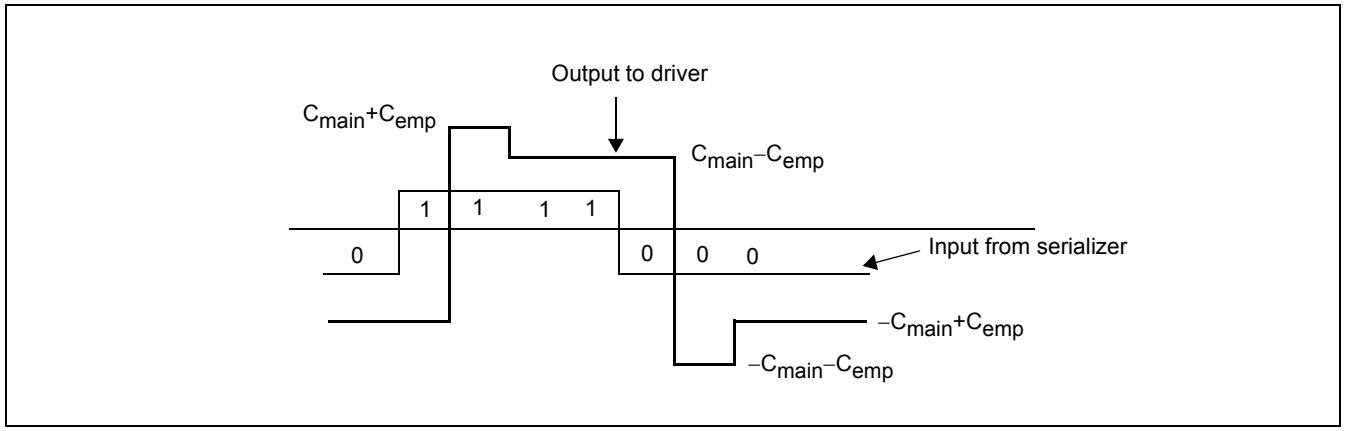
- Selectable de-emphasis and swing levels in the driver circuit
- Equalization at the receiver end

20.7.1 Programmable Driver Power Levels

The CML transmitter contains a PCI Express compliant de-emphasis driver. It includes a two tap FIR pre-equalizer that can be programmed using SDR0_PEn_LnDRV[DRV_LVL] and SDR0_PEn_LnDRV[EMP_POST] to produce normal PCI Express output drive level of greater than 800 mV p-p differential with a –3.5 dB de-emphasis. Additionally, the integrated, LC tank oscillator based PLL generates clean reference clocks and optimized output jitter performance.

To choose the approximate settings, use the following diagram:

Figure 20-4. Drive Level and Emphasis Waveform Guide



When choosing SDR0_PEn_LnDRV[DRV_LVL] and SDR0_PEn_LnDRV[EMP_POST], remember to use the single ended level columns. The maximum output swing achievable is approximately 850 mV.

The SerDes allows multiple drive settings. Additionally, there are multiple pre-emphasis settings possible. The user must realize that drive level and de-emphasis levels are programmed independently from each other. For instance, if the de-emphasis level is set to 175 mV and the driver is set to 650 mV, the waveform maximum level will be approximately 825 mV with a de-emphasis level of 475 mV. If, however, the user sets the driver to 400 mV, the de-emphasis level will remain 175 mV resulting in a maximum level of approximately 575 mV with a de-emphasis level of 225 mV.

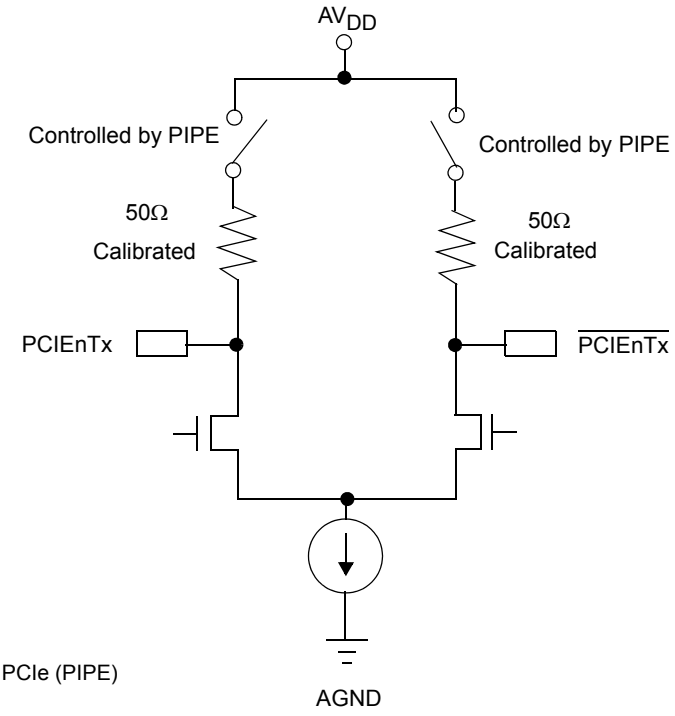
20.7.2 Transmitter-Receiver Diagram with Termination

The HSS-PCI Express function is designed for use in a doubly-terminated system (termination at driver end and receiver end). Both driver and receiver termination is included in the HSS and is calibrated to 50Ω single-ended (100Ω differential). Both transmitter and receiver can be set to high impedance according to the PCI Express 1.1 specification.

Systems should be AC coupled between transmitter and receiver.

The transmitter is terminated internally to AV_{DD} through 50Ω calibrated resistors as shown below in *Figure 20-5*. The PIPE logic selects whether these terminations are enabled according to the current power state. The termination resistors are calibrated at startup.

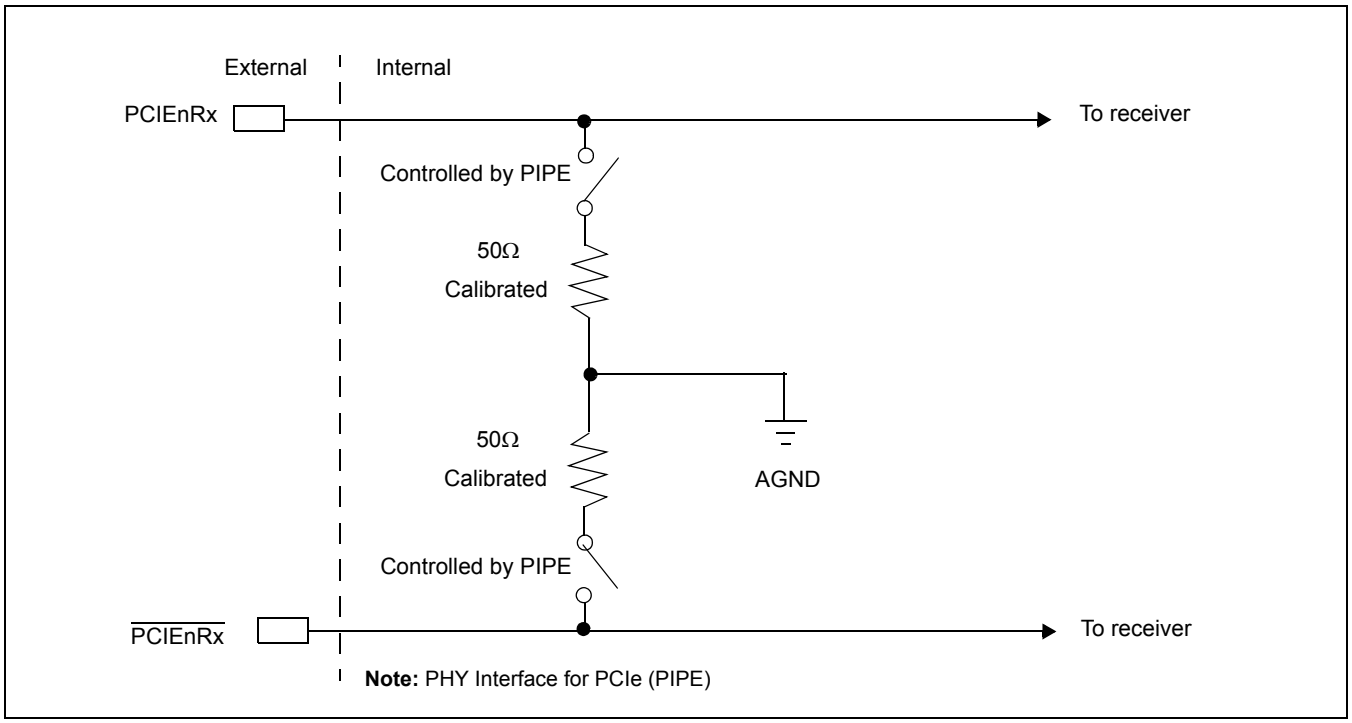
Figure 20-5. Transmit Termination



Note: PHY Interface for PCIe (PIPE)

The receiver is parallel terminated internally with calibrated 50Ω resistors (100Ω differential) as shown below in *Figure 20-6*. The common mode terminated to AGND as required by the PCI Express 1.1/1.0a specification. The termination is controlled by the PIPE logic.

Figure 20-6. Receiver Termination



20.8 PCI Express Power Management

The *PCI Express Specification* defines four power management (PM) states for devices (D0, D1, D2, and D3), and four PM states for links (L0, L0s, L1, and L2).

20.8.1 Device PM States

Device-Supported PCI Express Power States:

- D0 – Normal Operation: device fully functional, no power conservation needed
- D1 – Light sleep
- D2 – Deep sleep
- D3 – Device OFF: not running, context lost, device can be removed

20.8.2 Link PM States

Link-Supported PCI Express Power States:

- L0 – Normal operation: active transmit and receive
- L0s – Power saving: transmit and receive Idle
- L1 – Standby: transmit and receive in sleep mode
- L2 – Powered down: transmit and receive powered off, beacon enabled

20.8.3 Beacon Signaling

The beacon is a way to signal the upstream component that software needs to be notified of the wakeup request.

A PCI Express device that is in the L2 low power state can generate a wakeup event to notify the system that it wants to change to the full-on L0 state.

The beacon is a signaling function designed to operate on AUX power and does not require the use of the differential drivers and receivers. It is a low speed signal transmitted on lane 0.

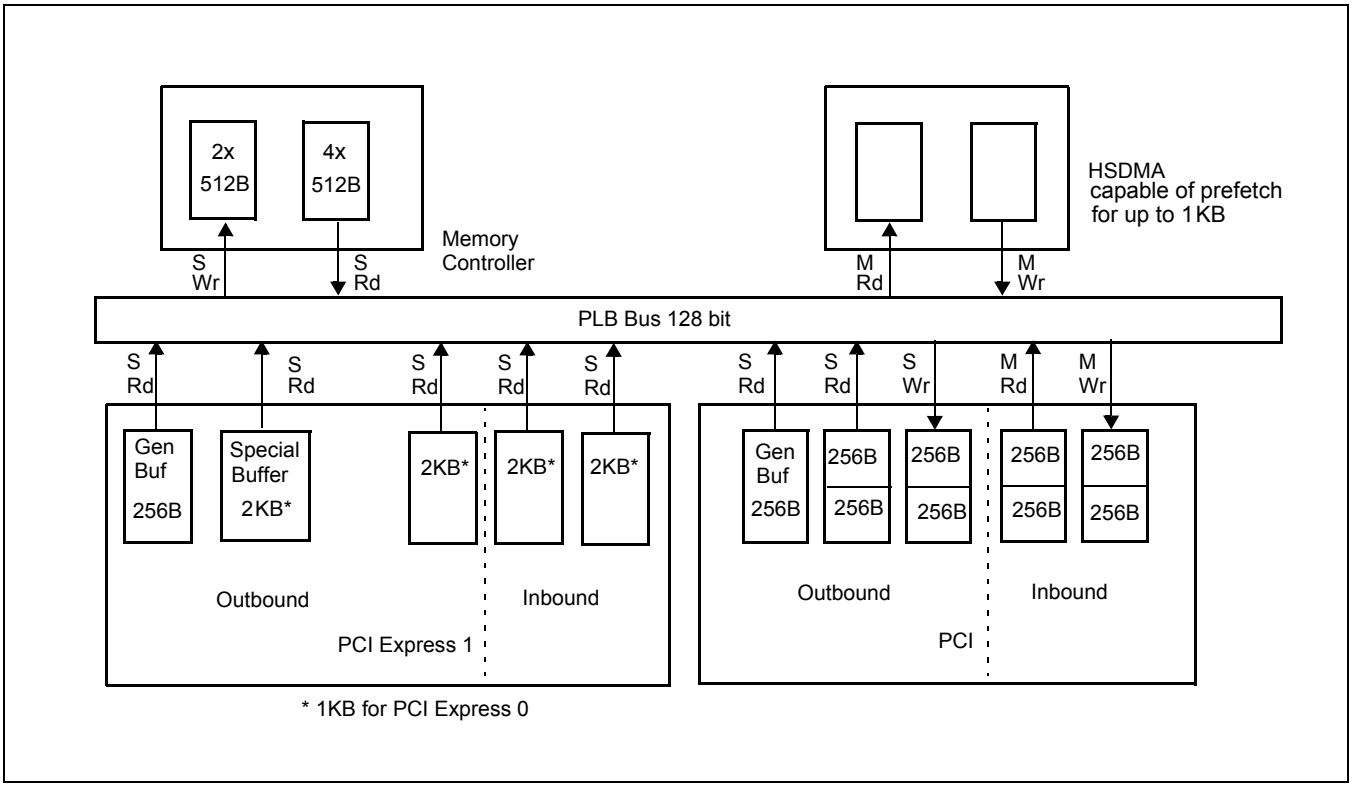
20.8.4 Data Buffering on the PLB Interface

Data can be read or written on both inbound and outbound transactions for each of the PCI Express ports. In order to improve performance for outbound reads, a special 2-KB buffer is provided, logically organized into four 512-B buffers to match the HSDMA 512-B maximum burst size. Each 512-B buffer has its own buffer handler. Prefetch mode is required to access these buffers. HSDMA is the only bus master that can operate in prefetch mode and generate prefetch attributes to PCI Express. Only one of the 512-B buffers is used.

The following figure shows the relationship between the PCI Express special buffers and the PLB bus. The PEGPLn_SPECIAL register is used to program the buffer handlers.

.

Figure 20-7. PCI Express Data Buffering on the PLB Interface



20.8.5 Glossary of Terms

ACK/NAK	Acknowledgement protocol for completion of a transaction
ACPI	Advanced Configuration and Power Interface
AER	PCI Express Advanced Error Reporting
AL	Application Layer

ASPM	Active State Power Management
CFG	Configuration
CRC	Cyclic Redundancy Checking
DL	Data Link
DLL	Data Link Layer
DLLP	Data Link Layer Packet
Downstream Port	Port that points away from the Root Complex
DSN	PCI Express Device Serial Number
DW	Double Word (32 bits)
EC	PCI Express capability
ECRC	End to End CRC
EMI	Electromagnetic Interference
Endpoint	A device with a Type 00h Configuration Space header
GBIF	Generic Bus Interface
HSS	High Speed Serializer/Deserializer
Inbound	Transaction from PCI Express outside to inside the PPC460EX/EXr/GT
Ingress Port	Port that receives a packet
Lane	Serial point-to-point Physical link that has one pair of Tx and one pair of Rx differential signals
Link	A PCI Express Physical link can be made with 1, 2, 4, 8, 16, or 32 lanes
LCRC	Data Link Layer CRC
MSI	PCI Message Signaled Interrupt
NDP	Normalized Driver Power
Non Posted	Write transaction completed at the originating bus after completion at the destination
Opaque	Non transparent bridge
Outbound	Transaction from inside the PPC460EX/EXr/GT to an outside PCI port
Outgress Port	Port that transmits a packet
PB	PCI Express power budgeting
PIPE	Physical Interface for PCI Express
PM	Power management
PME	Power management Event
Poisoned	TL packet that has been corrupted
Port	Each side of a Link has a Port
Posted	Write transaction completed at the originating bus before completion at the destination
QoS	Quality of Service
RCB	Read Completion Boundary
Replay Buffer	Buffer of TLPs in the transmitter that are resent in case of detected error at receiver side
Root Complex	An entity that includes a host bridge and one or more Root Ports
Scrambling	Each byte of a PCI packet is scrambled to reduce average EMI noise
Split Transaction	Completion separated from the Request
SYS	System
TCs	Traffic Classes
TL	Transaction Layer
TLP	Transaction Layer Packet
Transparent	Bridge in which primary and secondary busses share the same address space
UIC	Unified Interrupt Controller
Upstream Port	Port that is in the direction of the Root Complex
UREG	User defined registers (0–4 RO, 0–4 R/W registers)
UTL	Upper Transaction Layer
VC	PCI Express virtual channel
XBUS	Optional expansion bus
8b/10b	Encoding of 8 bits on 10 bits to include clock transitions in the signal

20.9 PCI Express Error Processing

The PCI Express architecture provides a rich set of error detection, reporting, and logging capabilities. PCI Express uses three classes of errors: correctable, non fatal, and fatal uncorrectable.

20.9.1 PCI Express Uncorrectable Errors

The following registers manage PCI Express uncorrectable errors for Endpoint and Root Port:

- PECFGn_AERUESTA AER Uncorrectable Error Status Register - 0x104
- PECFGn_AERUEMSK AER Uncorrectable Error Mask Register - 0x108
- PECFGn_AERUESEV AER Uncorrectable Error Severity Register - 0x10C

PCI Express handles the following uncorrectable errors:

- Data Link Protocol Error
- Poisoned TLP
- Flow Control Protocol Error
- Completion Timeout
- Completer Abort
- Unexpected Completion
- Receiver Overflow
- Malformed TLP
- ECRC Error
- Unsupported Request Error

20.9.2 PCI Express Correctable Errors

The following registers manage PCI Express correctable errors for Endpoint (EP) and Root Port (RP):

- PECFGn_ AERCESTA AER Correctable Error Status Register - 0x110
- PECFGn_AERCEMSK AER Correctable Error Mask Register - 0x114

PCI Express handles the following correctable errors:

- Replay Timer Timeout
- Replay NUM Rollover
- Bad DLLP
- Bad TLP
- Receiver Error

The following registers enable the processing of both correctable and uncorrectable errors, by means of the indicated register bits:

PECFGn_AERCAPCTL AER Capability and Control Register (EP and RP) - 0x118

- 8 AER - ECRC Check Enable [CHKE]
- 7 AER - ECRC Check Capable [CHKA]
- 6 AER - ECRC Generation Enable [GEN]
- 5 AER - ECRC Generation Capable [GENC]
- 4:0 AER - First Error Pointer Register [FERP]

PECFGn_AERRTCTL AER Root Error Control Register (RP only) - 0x12C

- 2 Fatal Error Reporting Enable [FERRE]
- 1 Non Fatal Error Reporting Enable [NFERE]
- 0 Correctable Error Reporting Enable [CERRE]

PECFGn_AERRTSTA AER Root Error Status Register (RP only) - 0x130
- 0 ERR_COR Received [RERRC]
- 1 Multiple ERR_COR Received [RMERC]
- 2 ERR_FATAL/NONFATAL Received [FNFER]
- 3 Multiple ERR_FATAL/NONFATAL Received [MULTI]
- 4 First Uncorrectable Fatal [FIRST]
- 5 Non Fatal Error Messages Received [RNFER]
- 6 Fatal Error Messages Received [RFERR]
- 27:31 Advanced Error Interrupt Message Number [MSIN]

PECFGn_AERERRSID AER Error Source Identification Register (RP only) - 0x134
- 31:16 ERR_FATAL/NONFATAL Source Identification [FNF]
- 15:0 ERR_COR Source Identification [CORR]

PECFGn_AERCAPCTLPA AER Capability and Control Programming Access Register (EP and RP) - 0x318
- 7 ECRC Check Capable programming access for PECFGn_AERCAPCTL[CHKA]
- 5 ECRC Generation Capable programming access for PECFGn_AERCAPCTL[GENC]

PECFGn_AERRTSTAPA AER Root Error Status Programming Access register (RP only) - 0x330
- 27:31 Advanced Error Interrupt Message Number programming access for PECFGn_AERRTSTA[MSIN]

20.9.3 PCI Express Error Status

The PCI Status and Command Registers are used to communicate PCI Express port status and control information between the hardware and software.

PECFGn_CMDSTATUS PCI Status and Command Registers - 0x004
- 31 Detected Parity Error [DEPE]: The PCI Express port has received a poisoned TLP.
- 30 Signaled System Error [SSE]: Set if the PCI Express port sends fatal or non fatal messages and the SERR Enable bit (PECFGn_CMDSTATUS[SERE]) is set to 1.
- 29 Received Master Abort [RMA]: The PCI Express port has received completion with an unsupported request.
- 28 Received Target Abort [RTA]: The PCI Express port has received Completion Abort.
- 27 Signaled Target Abort [STA]: The PCI Express port has signaled Completion Abort.
- 24 Master Data Parity Error [DPE]:The PCI Express port has signaled poisoned TLP.

The PCI Secondary Status Register (PECFGn_IOBASE) is similar in function and bit definition to the status portion of the PCI Status and Command Registers (PECFGn_CMDSTATUS). However, it reflects the status of the secondary bus, while PECFGn_CMDSTATUS reflects status of the primary bus.

PECFGn_IOBASE PCI Secondary Status, I/O Limit, and I/O Base Registers - 0x01C
- 31 Detected Parity Error [DEPE]: PCI Express received a Poisoned TLP on the Root Complex Device Secondary Bus.
- 30 Received System Error [SSE]: PCI Express received non fatal/fatal Error Message on the Root Complex Device Secondary Bus.
- 29 Received Master Abort [RMA]: PCI Express received Completion with Unsupported Request on the Root Complex Device Secondary Bus
- 28 Received Target Abort [RTA]: PCI Express received Completion Abort on the Root Complex Device Secondary Bus
- 27 Signaled Target Abort [STA]: PCI Express signaled Completion Abort on the Root Complex Device Secondary Bus
- 24 Master Data Parity Error [MDPE]:PCI Express Poisoned TLP on the TYPE1 Device Secondary Bus

Table 20-1. PCI Express Error Processing

PCI Express Base Spec. Rev. 1.0a Error Refs.		Who detects / responds? D = detecting agent, R = responding agent (Responding agent signals error to CFG and (in the case of the AL/UTL) translates CFG outputs to Root Complex-level events such as interrupts as appropriate)		
	Error Name	**DLP**	**TL**	**AL/UTL**
2.7.2.2	**Poisoned TLP Received** **Note:** Except as noted in the spec, the processing of Poisoned TLPs is implementation specific. The AL/UTL may decide to discard the received TLP or forward it, but must update Flow Control information normally in each case.			**DR** AL/UTL receiver detects EP bit set in header. AL/UTL discards TLP and updates receive Flow Control information normally in either case. AL/UTL signals error to CFG and provides header to CFG for logging. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.7.1	**ECRC Check Failed**		**D** TL receiver detects ECRC error and notifies AL/UTL. TL updates receive Flow Control information normally.	**R** AL/UTL discards TLP and updates receive Flow Control information normally. AL/UTL signals error to CFG and provides header to CFG for logging. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.2.2 2.2.3 2.2.7 2.2.8.1 2.2.8.2 2.2.8.3 2.2.8.4 2.2.8.5 2.2.9 2.3 2.3.1.1 2.5 2.5.3 2.6.1 6.3.2	**Malformed TLP** **Note:** Malformed TLPs are a broad set of errors. Some are detectable by TL, with the rest only detectable by AL/UTL. The bold entries in the far left column indicate spec. sections where there is a mix of TL detected and AL/UTL detected errors.		**D** TL receiver detects illegal TLP format and notifies AL/UTL of Malformed TLP. All TL detected Malformed TLP cases result in NO receive Flow Control information updates. All AL detected Malformed TLP cases result in normal receive Flow Control updates.	**R** AL/UTL discards TLP. All TL detected Malformed TLP cases result in NO receive Flow Control information updates. All AL detected Malformed TLP cases result in normal receive Flow Control information updates. AL/UTL signals error to CFG and provides header to CFG for logging. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.2.5 2.2.7 2.3.1 2.3.1.1 2.3.2	**Malformed TLP** **Note:** The bold entries in the far left column indicate spec. sections where there is a mix of TL detected and AL/UTL detected errors.			**DR** AL/UTL receiver detects illegal TLP format. AL/UTL discards TLP. All AL detected Malformed TLP cases result in normal receive Flow Control information updates. AL/UTL signals error to CFG and provides header to CFG for logging. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.

Table 20-1. PCI Express Error Processing (Continued)

PCI Express Base Spec. Rev. 1.0a Error Refs.		Who detects / responds? D = detecting agent, R = responding agent (Responding agent signals error to CFG and (in the case of the AL/UTL) translates CFG outputs to Root Complex-level events such as interrupts as appropriate)		
2.2.8.6 2.3.1 2.3.2 2.7.2.2 2.9.1 5.3.1 6.2.3.1 6.2.6 6.2.7.1 6.5.7 7.3.1 7.3.3 7.5.1.1 7.5.1.2	**Unsupported Request**			**DR** AL/UTL receiver receives a request that it determines to be unsupported. AL/UTL discards TLP and updates receive Flow Control information normally. If request requires a completion (i.e., NP), then AL/UTL sends a completion packet with UR completion status. AL/UTL signals error to CFG and provides header to CFG for logging. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.3.1	**Completer Abort**			**DR** AL/UTL receiver receives a request that it flags as Completer Abort. AL/UTL discards TLP and updates receive Flow Control information normally. If request requires a completion (that is, NP), then AL/UTL sends a completion packet with CA completion status. AL/UTL signals error to CFG and provides header to CFG for logging. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.3.2	**Unexpected Completion**			**DR** AL/UTL receiver receives an Unexpected Completion. AL/UTL discards TLP and updates receive Flow Control information normally. AL/UTL signals error to CFG and provides header to CFG for logging. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.6.1	**Flow Control Protocol Error**		**DR** TL receiver detects Flow Control Protocol Error and signals error to CFG	**R** CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.6.1.2	**Receiver Overflow**		**DR** TL receiver detects Receiver Overflow and signals error to AL/UTL. TL does not update receive Flow Control information.	**R** AL/UTL discards TLP and does not update receive Flow Control information. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
2.8	**Completion Timeout**			**DR** AL/UTL receiver detects a Completion Timeout. AL/UTL signals error to CFG. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.

Table 20-1. PCI Express Error Processing (Continued)

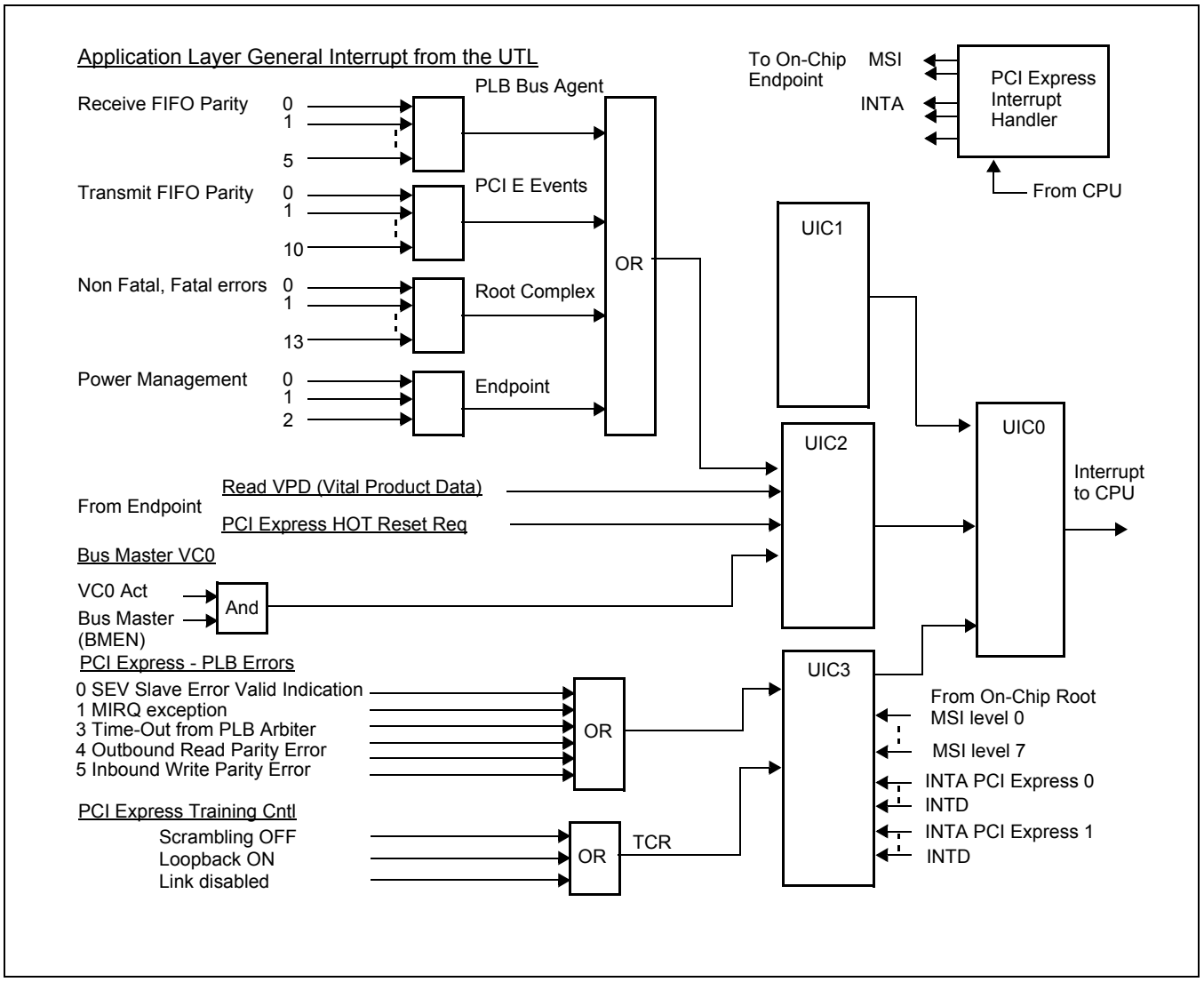
PCI Express Base Spec. Rev. 1.0a Error Refs.		Who detects / responds? D = detecting agent, R = responding agent (Responding agent signals error to CFG and (in the case of the AL/UTL) translates CFG outputs to Root Complex-level events such as interrupts as appropriate)		
3.5.2.1	**Replay NUM Rollover**	**DR** DLP detects Replay NUM Rollover. DLP initiates a Link retraining and signals error to CFG.		**R** CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
3.5.2.1	**Replay Timeout**	**DR** DLP detects Replay Timeout. DLP initiates a reply and signals error to CFG.		**R** CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
3.5.2.1	**Bad DLLP -LCRC**	**DR** DLP receiver detects DLLP CRC error. DLP discards DLLP and signals error to CFG		**R** CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
3.5.2.1	**Data Link Layer Protocol Error**	**DR** DLP receiver detects protocol error. DLP discards DLLP and signals error to CFG.		**R** CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
3.5.3.1	**Bad TLP - Seq. Number - LCRC**	**DR** DLP receiver detects sequence number or LCRC error. DLP signals "Nullify" to TL and AL/UTL (in LCRC error case only). DLP signals error to CFG.	**R** TL discards TLP and does not update receive Flow Control information.	**R** AL/UTL discards TLP and does not update receive Flow Control information. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.
4.2.1.3 4.2.2.1 4.2.4.4 4.2.6	**Receiver Error**	**DR** DLP receiver detects symbol or framing type errors. DLP retrains link in order to re-establish symbol lock. DLP signals "Nullify" to TL and AL/UTL. DLP signals Receiver Error to CFG.	**R** TL discards TLP and does not update receive Flow Control information.	**R** AL/UTL discards TLP and does not update receive Flow Control information. CFG manages configuration space status bit settings and if enabled outputs error indication (interrupt) for use by Root Complex.

20.10 PCI Express Internal Interrupts

The PCI Express ports can generate various interrupts, which correspond to specific events or errors occurring in the PCI Express circuitry. These interrupts are input to the chip's interrupt handler (UIC). For some interrupt requests, the sources of the interrupts are regrouped, which requires reading the corresponding status registers to find the root cause of the interrupt. All sources of interrupts can be disabled (masked).

Figure 20-8 shows the internal interrupt sources for PCI Express port 0 that are inputs to the UICs.

Figure 20-8. PCI Express Internal Interrupts



For PCI Express port 0 and 1, the interrupts that are input to the UIC3 described in this section are:

- IRQ0 and IRQ6 Application Layer General interrupt from the UTL
- IRQ1 and IRQ7 VPD Access: Read access of the Vital Product Data (VPD) by a Root Port
- IRQ2 and IRQ8 Start Hot Reset Request
- IRQ3 and IRQ9 Training Control Interface Receive
- IRQ4 and IRQ10 BusMaster VC0
- IRQ5 and IRQ11 PLB Error (DCR: Device Control Register)

20.10.1 IRQ0, IRQ6: Application Layer General Interrupt from the UTL

Interrupt input IRQ0 or IRQ6 for UIC3 is the logical OR of 34 interrupt sources. The status for these sources is in the UTL Status Register (PEUTLn_STA).

- 0 PBS PLB Bus Agent, 6 sources
- 1 PES PCI Express Port Event, 11 sources
- 2 RCES Root Complex Event, 14 sources
- 3 EPES Endpoint Event, 3 sources

UTL function native interrupt source resolution is implemented in two levels. To determine the interrupt source, the interrupt handler software must first read the UTL Status register (PEUTLn_STA) and then read the appropriate second-level status register (PEUTLn_PBASTA, PEUTLn_PSTA, PEUTLn_RCSTA, or PEUTLn_EPSTA).

Detection of events that can trigger the interrupt can be disabled by resetting the bit in the Error Enable Register that corresponds to the type of event (PEUTLn_PBAEEN, PEUTLn_PERREN, PEUTLn_RCERREN, or PEUTLn_EPERREN). The software can configure specific events not to trigger interrupt assertion by resetting the corresponding bit in the appropriate Interrupt Enable Register (PEUTLn_PBAIEN, PEUTLn_PIRQEN, PEUTLn_RCIRQEN, or PEUTLn_EPIRQEN).

PLB Bus Agent

The sources of PLB Bus Agent interrupts are controlled by the following registers:
- PEUTLn_PBASTA PLB Bus Agent Status Register
- PEUTLn_PBAEEN PLB Bus Agent Error Enable Register
- PEUTLn_PBAIEN PLB Bus Agent Interrupt Enable Register

These interrupt sources are:
- RNPPInbound Non Posted Request FIFO (RXNP) Parity error
- RFTPInbound Read Tag FIFO (RXFT) parity error
- RPCPInbound Posted Request FIFO (RXPC) parity error
- RCIPReceived Completion Indication FIFO (RXCIF) parity error
- RCCPReceived Completion Processing FIFO (RCC) parity error
- OWLMOutbound Memory Write Length Mismatch error. The data size specified by the outbound write request and the amount of data written by the PCI Express PLB Slave to the Tx Posted Data Buffer are different.

PCI Express Port Event

The sources of PCI Express Port Event interrupts are controlled by the following registers:
- PEUTLn_PSTAPCI Express Port Status Register
- PEUTLn_PERRENPCI Express Port Error Enable Register
- PEUTLn_PIRQENPCI Express Port Interrupt Enable Register

These interrupt sources are:
- TXDP Parity error detected by the Transaction Layer during outbound TLP transmission
- TPCPOutbound Posted Requests FIFO (TXPC) parity error
- TNPPOutbound Non Posted Requests FIFO (TXNP) parity error
- TFTPOutbound Non Posted Requests Free Tags FIFO (TXFT) parity error
- TCAP Transmit Completions Attribute array (TXCA) parity error
- TCIP Transmit Completions Indications FIFO (TXCIF) parity error
- ITHP Inbound TLP Header parity error
- PLUP PCI Express link changed to the Link-Up state
- PLDN PCI Express Link changed to the Link-Down state
- OTDS Outbound Request Discarded
- CIRReceive Credit Initialization Error (TL not enabled to initialize credits)

Root Complex Event

The sources of Root Complex Event interrupts are controlled by the following registers:
- PEUTLn_RCSTA Root Complex Status Register
- PEUTLn_RCERREN Root Complex Error Enable Register
- PEUTLn_ RCIRQEN Root Complex Interrupt Enable Register

These interrupt sources are:
- RLCS Correctable system error occurred in the Root Port device
- RLNS Local Non Fatal Error in the Root Port device
- RLFS Local Fatal Error in the Root Port device
- RRCS Correctable system error occurred in the remote PCI Express device

- RRNS Non fatal system error occurred in the remote PCI Express device
- RRFS Fatal system error occurred in the remote PCI Express device
- RCPS Power Management Event (PME) signaled by the remote PCI Express device
- RCTS Received PME Turn Off (PME_TO_Ack) message acknowledge
- PAAS Legacy INTA interrupt is asserted
- PABS Legacy INTB interrupt is asserted
- PACS Legacy INTC interrupt is asserted
- PADS Legacy INTD interrupt is asserted
- ALES ASPM L1 Request received by the DLLP
- CRSS Asserted when configuration request is completed by the destination with Configuration Retry Status (CRS)

Endpoint Event

The sources of Endpoint Event interrupts are controlled by the following registers:
- PEUTLn_EPSTA Endpoint Status Register
- PEUTLn_EPERREN Endpoint Error Enable Register
- PEUTLn_EPIRQEN Endpoint Interrupt Enable Register

These interrupt sources are:
- PTOM PME_Turn_Off message received
- PMCS PM state (PECFGn_PMCSR[STATE] changed in the configuration space.
- AENS ASPM L1 negotiation abort. Asserted if a PM_Active_State_Nak message is received from the upstream device.

20.10.2 IRQ1, IRQ7: VPD Access: Read Access of the Vital Product Data (VPD) by a Root Port

This interrupt is generated by an Endpoint when a Root Port performs a read access of the Vital Product Data (VPD) information located within the Endpoint (EP).

The interrupt is generated if the Root Port detects that the Root Port has performed a write access to PECFGn_VPDCAP[FLAG].

20.10.3 IRQ2, IRQ8: Hot Reset Request

A Root Port generates a Hot Reset request to the Endpoint by sending a packet with Training Sequence TS1 and TS2. An interrupt Hot Reset request is generated by the Endpoint after decoding it in the TS1/TS2 packet.

The end of Hot Reset Request is given by SDR0_PEn_RCSSTS[HRSTRQ]. When this bit returns to 0, the request is over.

20.10.4 IRQ3, IRQ9: Training Control Interface Receive (TCR)

The three commands inside the DLP are:
- Scrambling Off SDR0_PEn_DLPSET[SROFF]
- Loopback ON SDR0_PEn_DLPSET[LBACK]
- Link disable PECFGn_ECLNKCTL[DISA]

20.10.5 IRQ4, IRQ10: BusMaster VC0 (Virtual Channel 0)

This interrupt occurs if the link is no longer active (VC0 goes down). See *PCI Express Port Reset Sequences* on page 516 for details.

The interrupt is the AND of the Bus Master signal from CFG and the VC0ACT signal from the Transaction Layer (TL) function.

- Bus Master signal of the Configuration (CFG) function)

  Normal link operation is when SDR0_PEn_RCSSTS[BMEN] = 1, which occurs when initialization and configuration of the link is complete. This bit is a copy of PECFGn_CMDSTATUS[PME].

- VC0ACT signal from the TL function

  Normal operation is when SDR0_PEn_RCSSTS[VC0ACT] = 1, which occurs when training and Flow Control initialization are complete.

VC0 Control can go down if a problem is detected in the TL.

Example of Usage of Interrupt 4 or 10: Link Going Down

If a link goes down, the PCI Express port detects that it can no longer exchange data, and the VC0 signal goes down, which sets IRQ4 or IRQ10 (if RQ4 or IRQ10 is programmed to turn on the falling edge).

The same interrupt input can be programmed to turn on the rising edge, in which case it can be used to signal the PPC440 processor when the configuration of the link is completed.

20.10.6 IRQ5, IRQ11: PLB Error (DCR: Device Control Register)

This interrupt is generated if it is enabled by PEGPLn_ESR[IE] and one of the following fields in that register is set to 1:

- SEV Slave Error Valid: When asserted, the Slave Error Address and Attribute Registers contain valid information.
- MIRQ MIRQ Received: MIRQ exception feedback from any PLB Slave to the PCI Express PLB Master interface.
- MERR MERR Received: Transaction error (Read and Write) indication feedback from any PLB Slave during a transaction initiated by the PCI Express PLB Master interface.
- TOT Time-Out Received: Time-Out indication feedback from the PLB Arbiter during a transaction initiated by the PCI Express PLB Master interface.
- ORPE Outbound Read Parity Error.
- IWPE Inbound Write Parity Error.

20.11 Messaging

PCI Express supports the following types of transactions, corresponding to independent address regions on the PLB:

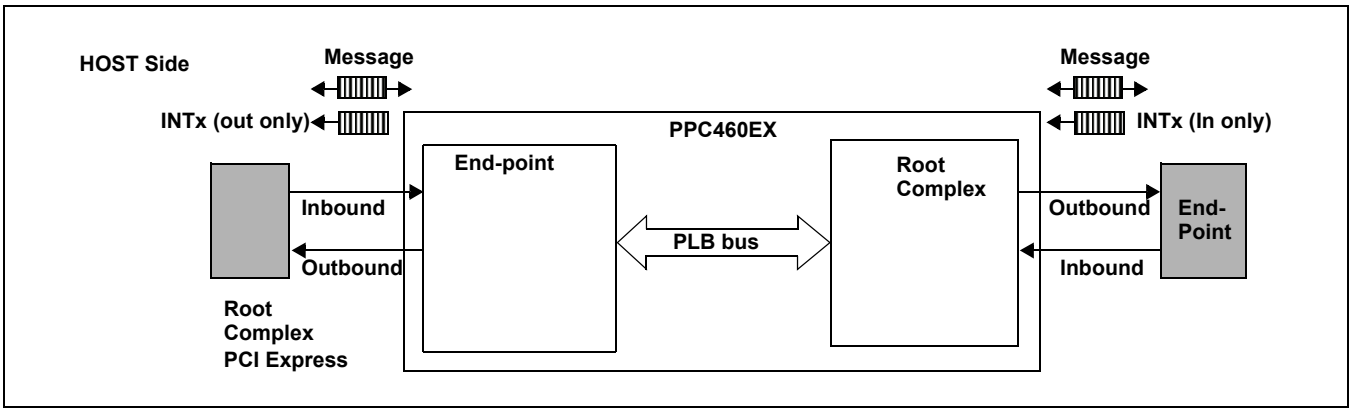
- Memory
- I/O
- Message
- Configuration
- Register

The regions have programmable addresses as described in *Table 20-2*.

Table 20-2. PCI Express Transactions

Region		Outbound		Inbound
		PLB address	PCI Express Address	
Memory	MSI	BAR0 BAR1	POM0 POM1	
I/O		BAR2	POM2	n/a
Message	INTx	BAR3	POM3	
Configuration		BAR4	POM4	
Register		BAR5	n/a	

Figure 20-9. PCI Express Messaging Capabilities



20.11.1 Message Support in Root Complex Applications

This section describes the message support in Root Complex applications.

Supported Inbound Messages – Root Complex Applications

Table 20-3 lists the supported inbound messages for Root Complex applications.

Table 20-3. Supported Inbound Messages – Root Complex Applications

Message Type	Messages Group	Data Payload	Message Processing
Assert_INTA	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PAAS] to 1.
Assert_INTB	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PABS] to 1.
Assert_INTC	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PACS] to 1.
Assert_INTD	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PADS] to 1.
Deassert_INTA	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PAAS] to 0.
Deassert_INTB	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PABS] to 0.
Deassert_INTC	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PACS] to 0.

Table 20-3. Supported Inbound Messages – Root Complex Applications

Message Type	Messages Group	Data Payload	Message Processing
Deassert_INTD	INTx Mechanism Messages	N	Set PEUTLn_RCSTA[PADS] to 0.
ERR_FATAL	Error Messages	N	Report to the CFG function.
ERR_NONFATAL	Error Messages	N	Report to the CFG function.
ERR_COR	Error Messages	N	Report to the CFG function.
PM_PME	PM Messages	N	Set PEUTLn_RCSTA[RCPS] to 1, report to the CFG function.
PME_TO_Ack	PM Messages	N	Set PEUTLn_RCSTA[RCTS] to 1.
Attention_Button_Pressed	Hot-Plug Signaling	N	Report to the CFG function.

INTx Processing – Root Complex Applications

The UTL function provides support for the legacy PCI INTx level interrupt mechanism by processing inbound Assert_INTx and Deassert_INTx messages and updating AL_INTx outputs. The UTL resets AL_INTx lines upon system reset or PCI Express link failure. The application may choose to configure UTL to set UTL interrupt line assertion in case of INTx interrupt activation instead of using AL_INTx interrupt lines.

Inbound Error Messages Processing – Root Complex Applications

The following types of inbound error messages correspond to the three types of PCI Express errors:
ERR_FATAL – Uncorrectable fatal error
ERR_NONFATAL – Uncorrectable non fatal error
ERR_COR – Correctable error

Error message type and relevant header fields are forwarded to the PCI Express CFG function for error reporting and logging.

Inbound Power Management Messages – Root Complex Applications

The UTL supports the following inbound power management messages, generated by the PCI Express Endpoints:
- PM_PME
- PME_TO_Ack

The UTL function reports reception of the PM_PME message to the PCI Express CFG function, which sets PECFGn_ECRTSTA[PMES] to 1 and stores the requestor ID of the device that triggered the PME message in PECFGn_ECRTSTA[PMEID]. See *EC Root Status Register (PECFGn_ECRTSTA)* on page 614 for details.

If a PME_TO_Ack message is received, the UTL function sets PEUTLn_RCSTA[RCTS] to 1. See *Root Complex Status Register (PEUTLn_RCSTA)* on page 578 for details.

Inbound Hot-Plug Signaling Messages – Root Complex Applications

The UTL supports inbound Attention_Button_Pressed message processing. This message is generated by the Endpoint device on the remote side of the PCI Express link if an Attention Button Pressed event is detected. The UTL function reports Attention_Button_Pressed message reception to the PCI Express CFG function, which sets PECFGn_ECSLTCTL[ATBP] to 1. See *EC Status and Slot Control Register (PECFGn_ECSLTCTL)* on page 612 for details.

Inbound MSI Processing – Root Complex Applications

PCI Express Message Signaled Interrupts (MSIs) appear as posted write transactions with a single DWord data payload. The UTL function converts inbound remote MSI writes to system bus master writes and presents them to the system bus master agent.

The UTL function does not support triggering of system bus master writes as a result of UTL internal events in a Root Complex configuration.

Supported Outbound Messages – Root Complex Applications

The UTL function supports outbound message generation according to PCI Express Root Complex message requirements. Messages issued over the system bus through the outbound write GBIF (write channel 9) are identified by the transaction's address attribute, bits A[52:59]. Address bits A[60:63] must always be zero for outbound messages. *Table 20-4* defines the messages supported by the UTL function in a Root Complex configuration.

Table 20-4. Outbound Messages – Root Complex Applications

Message Type	Messages Group	Data Payload	Message Generation
PME_Turn_Off	PM Messages	N	Outbound GBIF write, GBIF write channel 9, A[52:59]=0x19
PM_Active_State_Nak		N	Generated automatically by the UTL)
Set_Slot_Power_Limit	Slot Power Limit Support	Y	• Triggered by CFG_EC14__SLOT_PWR_SENDMSG indication from CFG, generated automatically by the UTL • Generated automatically after connection over the PCI Express link is established (DLP layer transitions from non Dl_up Status to DL_up Status)
Attention_Indicator_On	Hot-Plug Signaling	N	Triggered by CFG_EC18_ATN_IND_SENDMSG indication from CFG, generated automatically by the UTL
Attention_Indicator_Off		N	
Attention_Indicator_Blink		N	
Power_Indicator_On		N	Triggered by CFG_EC18_PWR_IND_SENDMSG indication from CFG, generated automatically by the UTL
Power_Indicator_Blink		N	
Power_Indicator_Off		N	

20.11.2 Message Support in Endpoint Applications

This section describes the message support in Endpoint applications.

Supported Inbound Messages – Endpoint Applications

Table 20-5 lists the supported inbound messages for Endpoint applications.

Table 20-5. Supported Inbound Messages – Endpoint Applications

Message Type	Messages Group	Data Payload	Message Processing	Message Routing Type
PME_Turn_Off	PM Messages	N	Set PEUTLn_EPSTA[PTOM] to 1.	Broadcast from Root Complex
PM_Active_State_Nak	PM Messages	N	Set PEUTLn_EPSTA[AENS] to 1.	Local - Terminate at receiver
Set_Slot_Power_Limit	Slot Power Limit Support	Y	Report to the CFG function.	Local - Terminate at receiver
Attention_Indicator_On	Hot-Plug Signaling	N	Report to the CFG function.	Local - Terminate at receiver
Attention_Indicator_Off	Hot-Plug Signaling	N	Report to the CFG function.	Local - Terminate at receiver
Attention_Indicator_Blink	Hot-Plug Signaling	N	Report to the CFG function.	Local - Terminate at receiver
Power_Indicator_On	Hot-Plug Signaling	N	Report to the CFG function.	Local - Terminate at receiver
Power_Indicator_Blink	Hot-Plug Signaling	N	Report to the CFG function.	Local - Terminate at receiver

Table 20-5. Supported Inbound Messages – Endpoint Applications

Message Type	Messages Group	Data Payload	Message Processing	Message Routing Type
Power_Indicator_Off	Hot-Plug Signaling	N	Report to the CFG function.	Local - Terminate at receiver

Outbound Messages – Endpoint Applications

The UTL function supports outbound message generation according to PCI Express Endpoint message requirements. Messages issued over the system bus through the outbound write GBIF (write channel 9) are identified by the transaction's address attribute, bits A[52:59]. Address bits A[60:63] must always be zero for outbound messages. *Table 20-6* defines the messages supported by the UTL function in an Endpoint configuration.

Table 20-6. Outbound Messages – Endpoint Applications

Message Type	Messages Group	Data Payload	Message Generation
ERR_FATAL	Error Messages	N	Fatal error Indication from CFG function
ERR_NONFATAL	Error Messages	N	Non fatal error Indication from CFG function.
ERR_COR	Error Messages	N	Correctable error Indication from CFG function.
Assert_INTA	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x20 External interrupts generator interface
Assert_INTB	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x21.
Assert_INTC	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x22.
Assert_INTD	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x23.
Deassert_INTA	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x24.
Deassert_INTB	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x25.
Deassert_INTC	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x26.
Deassert_INTD	INTx Mechanism Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x27.
PM_PME	PM Messages	N	Outbound GBIF write channel 9, A[52:59] = 0x18.
PME_TO_Ack	PM Messages	N	Generated automatically by the UTL.
Attention_Button_Pressed	Hot-Plug Signaling	N	Triggered by CFG_EC18_ATN_BTN_PRESSED indication from CFG, generated automatically by the UTL.

20.11.3 Messages with the I2O DMA Unit

Messages that do not follow PCI Express protocol can be transported on the PCI Express links between the PPC460EX/EXr/GT and the Host by using the I2O/DMA unit. For I2O Messaging using the I2O/DMA unit, the PCI Express port on the PPC460EX/EXr/GT is set as an Endpoint, while the port in the Host is Root Complex. Messages are stored either in PPC460EX/EXr/GT memory or in the Host memory. They are transported on Memory Write operations. The Message size is programmable in steps of 16-B blocks in the I2O MFA Size Count 0–7 (I2O0_MFAC0:7) registers to a maximum of 2^{32}. Its typical value is 64 bytes. Inbound (from the Host to the PPC460EX/EXr/GT) and outbound (from the PPC460EX/EXr/GT to the Host) messages are possible. The following types of I2O message transactions are possible:

- Inbound Pull Mode I2O responsible for the transaction by means of the associated DMA engine
- Inbound Push Mode: Host responsible for the transaction
- Outbound Push Mode:

20.11.3.1 Inbound Doorbells

If the Inbound Doorbell Register is written by an external PCI agent, an interrupt may be generated to the PPC440 processor. An interrupt is generated if any bit in the doorbell register is written to a value of 1.

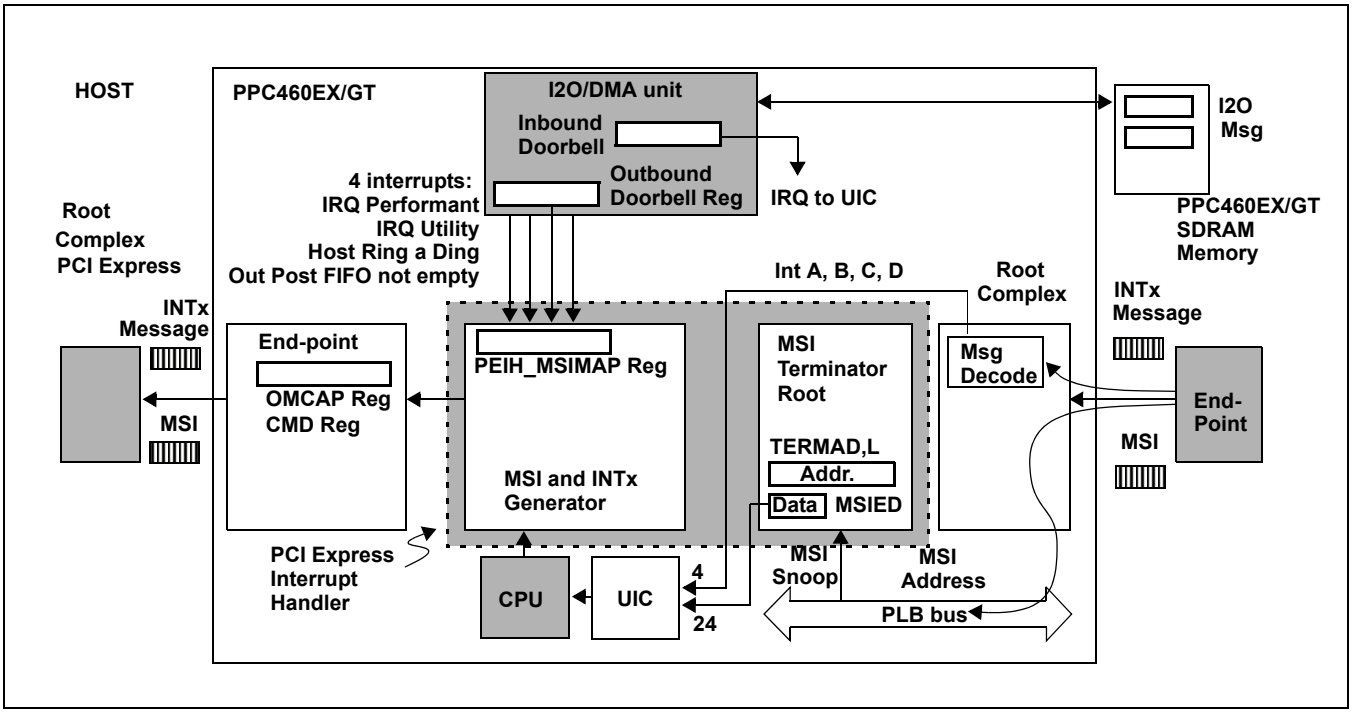
20.11.3.2 Outbound Doorbells

If the Outbound Doorbell Register is written by the PPC440 processor, an interrupt may be generated by means of the PCI Express Interrupt Handler as a PCI Express Message with $\overline{IRQ}$. If MSI is enabled, an MSI interrupt is generated. An interrupt is generated if any bit in the doorbell register is written to a value of 1. Once a bit is set in the Outbound Doorbell Register, it can be cleared by the Host.

20.12 PCI Express INTx and MSI Interrupt Handler

The PCI Express interrupt handler is a PLB Slave device attached to the Low Latency PLB segment. It handles PCI Express protocol stacks, generates INTx interrupts, and generates and terminates Message Signaled Interrupts (MSIs). It also enables the software to invalidate the outbound read completion general buffers of the PCI Express ports (see *Figure 20-1* on page 482 for details). *Figure 20-10* shows the PCI Express interrupt processing components.

Figure 20-10. PCI Express INTx and MSI Interrupt Processing



20.12.1 Message Signaled Interrupt (MSI)

MSIs are delivered to the Root Complex by performing memory write transactions. The MSI Capability Structure Registers provide all the information that the device requires to signal MSIs. These registers are set up by configuration software and include:

- Target memory address (PECFGn_OMMAL and PECFGn_OMMAU)
- Data to be written to the specific address location (PECFGn_OMMDATA)
- The number of messages that can be encoded into the data (PECFGn_OMCAP)

20.12.2 Features:

The PCI Express ports of the PPC460EX/EXr/GT support the following features:

Generation of A/B/C/D INTx Types for Endpoints.

- A/B/C/D type is programmable for individual PCI Express ports
- INTx interrupts are level sensitive, active high

Generation of MSIs for Endpoints

- Supports the generation of MSIs to any of the PCI Express stacks
- Does not support the generation of MSIs to multiple stacks simultaneously (only one stack can be configured as Endpoint)
- Supports one fixed 64-bit MSI address
- MSIs are generated on the rising edge of the interrupt line
- Supports the generation of up to four MSI messages
- MSI message requests can be asserted to any PCI Express stack
- Supports software mapping of any interrupt input to any enabled MSI message
- Permits the software to reallocate interrupts if the number of MSIs allocated is less than the number requested
- Any of the MSIs can be generated under program control by writing to a register
- Acknowledgment pin back to the I2O/DMA function ensures that MSIs are queued for transmission

MSI and MSI-X Termination for Root Ports

- Supports one fixed 64-bit MSI address. All Root Port stacks must be programmed with the same MSI address
- Supports up to 32 fixed MSI messages shared between the three PCI Express ports
- Provides one interrupt line per MSI (total of 32 interrupts) wired to the Universal Interrupt Controller (UIC)

20.12.3 Generating INTx Interrupts

INTx interrupt generation is enabled if PECFGn_CMDSTATUS[INTD] is set to 1, SDR0_PEn_RCSSTS[VC0ACT] is set to 1, and SDR0_PEn_RCSSTS[BMEN] is set to 1.

At interrupt assertion (rising edge), INT_GEN_REQ is asserted. INT_ADDR[61:63] defines the type of event to generate (Assert_INTx in this case). INTx_Status output is also asserted. The PCI Express stack acknowledges that the INTx message is queued for transmission by asserting INT_GEN_ACK.

At interrupt de-assertion (falling edge), INT_GEN_REQ is asserted. INT_ADDR[61:63] defines the type of event to generate (Deassert_INTx in this case). INTx_Status output is also de-asserted. The PCI Express stack acknowledges that the INTx message is queued for transmission by asserting INT_GEN_ACK.

If INTx generation becomes enabled while one of the interrupt inputs is asserted, the interrupt handler logic generates an Assert_INTx request and asserts INTx_Status.

20.12.4 Generating an MSI

MSI generation is enabled if PECFGn_OMCAP[MSIE] = 1, SDR0_PE0_RCSSTS[VC0ACT] = 1, and SDR0_PE0_RCSSTS[BMEN] = 1.

PECFGn_OMCAP[MMEN] defines the number of MSIs that can be generated:
- 1 MSI: MSI data field provided by the PCI configuration space remains unchanged.
- 2 MSIs: PCI configuration space data field bit 15 is overwritten. The value of bit 15 depends on interrupt type and MSI mapping.
- 4 MSIs: PCI configuration space data field bits 14:15 are overwritten. The value of bits 14:15 depends on the interrupt type and MSI mapping.

The interrupt handler logic detects the rising edge of the interrupt lines.

Note: If an interrupt line is asserted while a request is pending (INT_GEN_REQ is still asserted), the interrupt handler waits for INT_GEN_ACK assertion before it generates the next MSI.

Upon receiving INT_GEN_ACK, the interrupt handler forwards the acknowledgement to the DMA.

20.12.5 Terminating an MSI

The interrupt handler's PLB interface supports DWord memory writes. For these operations, the 64-bit PLB address is compared to the three termination address registers (two least significant bits are compared to 0).

If a match occurs, the interrupt handler performs the following operation:

(PLB_Data[0:26] XOR MSI_Expected_Data[0:26]) AND MSI_Mask[0:26]).

If the result is zero, a single PLB clock cycle pulse is generated to the UIC depending on the contents of PLB_Data[27:31].

PLB_Data[27:31] = 00000 → Interrupt bit 0
PLB_Data[27:31] = 00001 → Interrupt bit 1
PLB_Data[27:31] = ... → Interrupt bit n
PLB_Data[27:31] = 11111 → Interrupt bit 31

20.12.6 PCI Express Interrupt Handler Registers

The registers listed in *Table 20-7* are defined on the PLB bus from starting address 0xC 1000 0000. They are used to program the PCI Express interrupt handler.

Table 20-7. PCI Express Interrupt Handler Registers

Mnemonic	Register	Address	Access	Reset Value	Page
PEIH_TERMADH	PCI Express Termination Address High	0x00	R/W	0xFFFFFFFF	507
PEIH_TERMADL	PCI Express Termination Address Low	0x08	R/W	0xFFFFFFFC	507
PEIH_MSIED	MSI Expected Data	0x10	R/W	0x00000000	507
PEIH_MSIMK	MSI Mask	0x18	R/W	0x00000000	507
PEIH_MSIASS	Software Assert MSI	0x20	R/W	0x00000000	508
PEIH_MSIMAP	I2O MSI Mapping	0x28	R/W	0x00000000	508

Table 20-7. PCI Express Interrupt Handler Registers (Continued)

Mnemonic	Register	Address	Access	Reset Value	Page
PEIH_FLUSH0	PCI Express General Buffer Flush 0	0x30	Read only	0x00000000	508
PEIH_FLUSH1	PCI Express General Buffer Flush 1	0x38	Read only	0x00000000	508
	Reserved	0x40			
PEIH_CNTRST	PCI Express Counter Reset	0x48	Read only	0x00000000	508

PCI Express Termination Address High Register (PEIH_TERMADH)

Figure 20-11. PCI Express Termination Address High Register (PEIH_TERMADH)

0:31	TADH	PCI Express Termination Address High [0:31]	A DWord write to this address causes the logic to decode the MSI data field. All Root Port stacks must be programmed with the same MSI address.

PCI Express Termination Address Low Register (PEIH_TERMADL)

Figure 20-12. PCI Express Termination Address Low Register (PEIH_TERMADL)

0:29	TADL	PCI Express Termination Address Low	A DWord write to this address causes the logic to decode the MSI data field. All Root Port stacks must be programmed with the same MSI address.
30:31		Reserved	

MSI Expected Data Register (PEIH_MSIED)

Figure 20-13. MSI Expected Data Register (PEIH_MSIED)

0:26	MEDATA	MSI Expected Data	Expected most significant bits of the MSI data field. All Root Port stacks must be programmed with the same MSI data field.
27:31		Reserved	

MSI Mask Register (PEIH_MSIMK)

Figure 20-14. MSI Mask Register (PEIH_MSIMK)

0:26	MASK	MSI Mask	Defines the bits that need to be compared. A 1 value indicates that the bit is valid
27:31		Reserved	

Software Assert MSI Register (PEIH_MSIASS)

Figure 20-15. Software Assert MSI Register (PEIH_MSIASS)			
0:2	SOFTM	Software_Assert_MSI: 000 No MSI generated 001 MSI 0 generated. MSI Data field [14:15] = 00 010 MSI 1 generated. MSI Data field [14:15] = 01 011 MSI 2 generated. MSI Data field [14:15] = 10 100 MSI 3 generated. MSI Data field [14:15] = 11 Others: Reserved	Enables the software to generate any MSI. This field is cleared when the MSI is queued for transmission.
3:31		Reserved	

I2O MSI Mapping Register (PEIH_MSIMAP)

Figure 20-16. I2O MSI Mapping Register (PEIH_MSIMAP)			
0	PERFIE	I2O Performance Mode Completion Interrupt Enable	
1:2	PERFM	I2O Performance Interrupt MSI Number	
3	RINGIE	I2O Host_RingADing Interrupt Enable	
4:5	RINGM	I2O Host_RingADing MSI Number	
6	FIFOIE	I2O OBPL_FIFO_NE Interrupt Enable	(Outbound Post List FIFO Not Empty)
7:8	FIFOM	I2O OBPL_FIFO_NE MSI Number	
9	UTILIE	I2O Utility Interrupt Enable	
10:11	UTILM	I2O Utility MSI Number	
12:31		Reserved	

PCI Express General Buffer Flush n Register (PEIH_FLUSHn)

There are three General Buffer Flush registers (PEIH_FLUSH*n*, where *n* = 0:1), one for each PCI Express port—each with the same function.

Figure 20-17. PCI Express General Buffer Flush n Register (PEIH_FLUSHn)			
0:31	PEFn	PCI Express *n* Flush	A read to this register invalidates the contents of the general buffer. Returned data is all zeroes.

PCI Express Counter Reset Register (PEIH_CNTRST)

This register is read only.

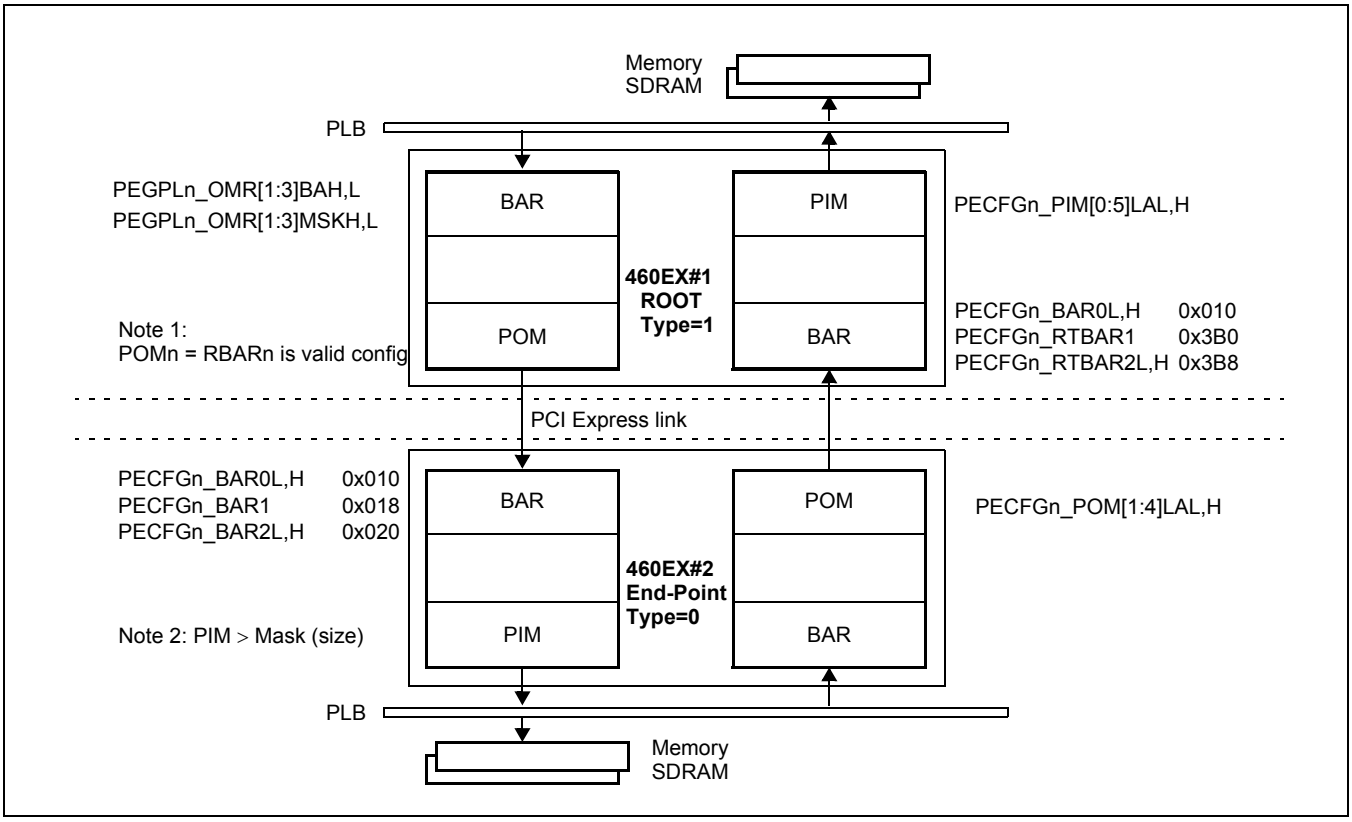
Figure 20-18. PCI Express Counter Reset Register (PEIH_CNTRST)			
0:31	COUNT		

20.13 PCI Address Translation

Each PCI Express port supports several address ranges, also called regions, in both PLB and PCI address spaces.

The registers involved in address translation for both inbound and outbound transactions for a PCI Express Endpoint or a Root Port are shown in *Figure 20-19* for memory access from a remote device.

Figure 20-19. Root and Endpoint Address Translation for Memory Access



20.13.1 Outbound Address Mapping

The PCI Express PLB Slave interface supports the following independent address regions on the PLB:

- Memory region 1 for outbound memory requests—a wide memory area allocation up to 2^{63} bytes. It is only enabled if PEGPLn_OMR1MSKL[UOT] is set.
- Memory region 2 for outbound memory requests.
- Region 3 can be configured as either a memory region or an I/O region by means of PEGPLn_OMR3MSKL[IO]. If used as an I/O region (PEGPLn_OMR3MSKL[IO] is set), transactions that involve this region are always treated as non posted transactions.
- Message region for message transactions.
- Configuration region for outbound configuration requests. In order to support both internal and external configuration transactions, the configuration address region size must be set to a minimum of 2^{29}. The PCI Express port examines PLB address bit 35 and, based on its setting, determines whether to issue a local or a remote configuration transaction.

- Register region for local registers (PEUTLn_xxxx).

The size of each of these regions is programmable as described in *Table 20-8*:

Table 20-8. Outbound Memory Ranges

Region	PLB Address	PCI Address	Size (Bytes)	Address Boundary	Posted or Non Posted	Access
Memory #1	BAR0 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	POM0 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{27} (128MB) to 2^{63}	N × Size	Posted	R/W
Memory #2	BAR1 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	POM1 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{20} (1MB) to 2^{63}	N × Size	Posted	R/W
Memory #3	BAR2 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	POM2 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{7} (128B) to 2^{63}	N × Size	Posted	R/W
I/O					Non Posted	
Message	BAR3 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	POM3 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{7} (128B) to 2^{15} (32KB)	N × Size	Posted	W
Configuration	BAR4 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	POM4 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{29} (512MB) to 2^{32} (4GB)	N × Size	Non Posted	R/W
Register	BAR5 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	NA	2^{7} (128B) to 2^{15} (32KB)	N × Size	NA	R/W

Notes:

1. All address regions are programmable in locations within the PLB address space at a granularity of their size. For example, if the size is 4 GB, then it begins on a 4-GB boundary.
2. Each region can be disabled or enabled using the VAL field defined in the mask register set.
3. For I/O, configuration, and register regions, only single-beat reads or writes are supported.

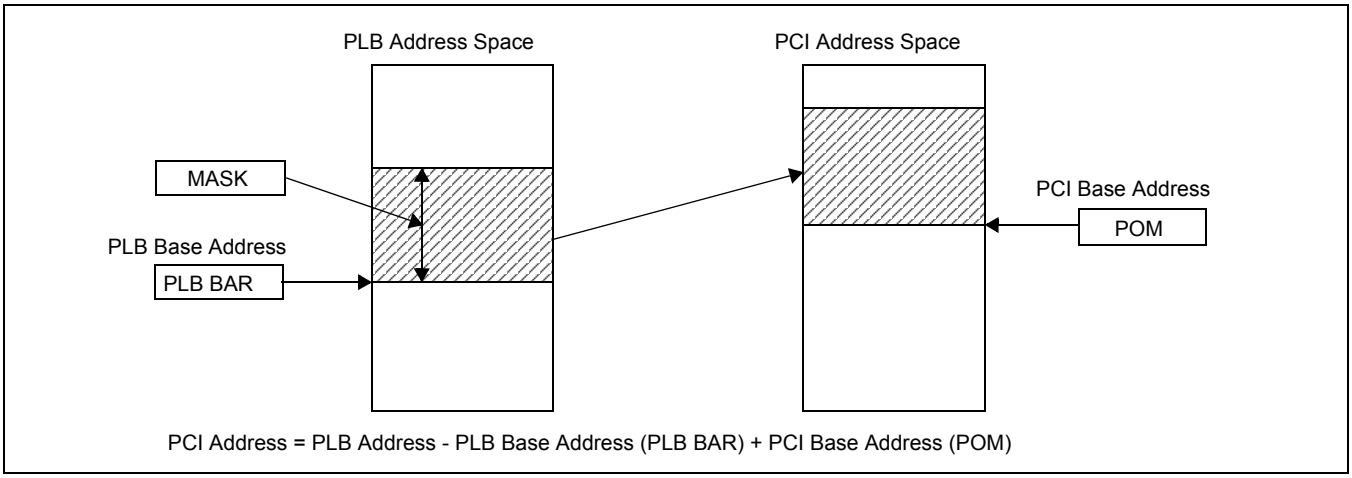
Typically, the configuration of these regions is changed only during the initialization process, not during chip operation. The registers that define these regions must only be changed during reset, or after reset *and* when none of the regions are enabled.

20.13.2 Outbound Address Translation

For address translation, each address region except the last one has an associated PCI base address vector, known as the PCI Outbound Map (POM). The PLB-to-PCI Express address translation function is performed by concatenating the MSBs of the POM vector to the LSBs of the incoming PLB address. The number of MSBs replaced is defined by the mask vector, which defines the size of the region. Only the msbs that are masked (set in the mask vector) are taken from the POM vector. The remaining PLB address bits (lsbs) are unchanged. Using the POM vector, the PLB Slave interface claims a region within the PCI Express address space.

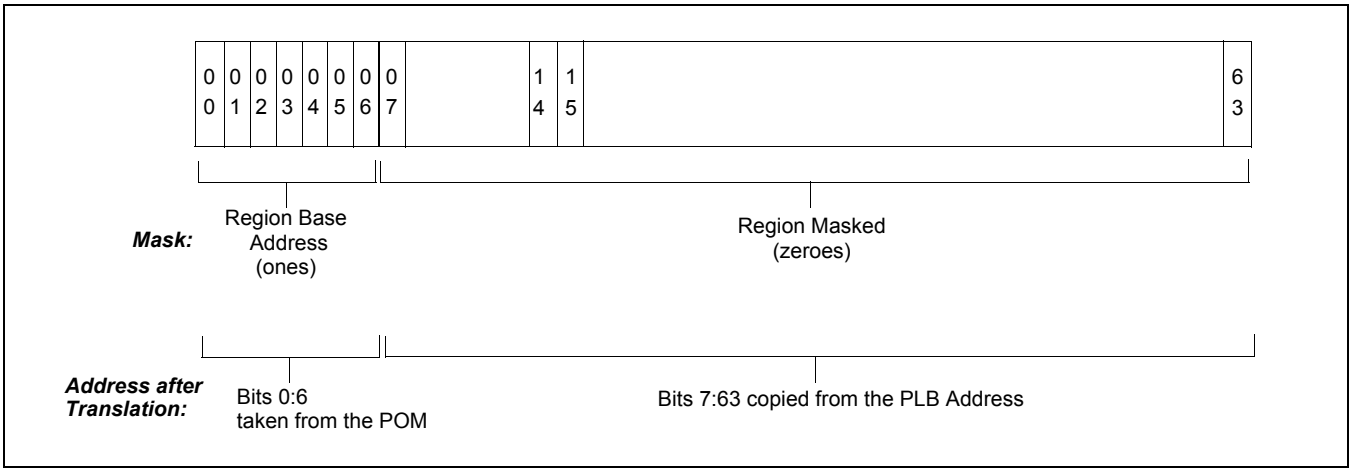
Figure 20-20 shows the outbound address translation logic. The PLB BAR vector defines a PLB address space offset. The POM vector defines a PCI Express address space offset. The size of both regions is identical and determined by the mask vector. The address translation function replaces the PLB space offset with the PCI space offset by replacing the MSBs from the PLB BAR vector with the MSBs from the POM vector, leaving all LSBs unchanged.

Figure 20-20. Outbound Address Translation Logic



For example, assume that the size of the memory region is 2^{57}. The mask vector is set so that bits 0:6 are ones and bits 7:63 are zeroes. The PLB Base Address (PLB BAR) of the region is set by PLB address bits 0:6. Thus, the PCI Express address is a concatenation of bits 0:6 from the POM input to bits 7:63 from the PLB address, as shown in *Figure 20-21*.

Figure 20-21. Outbound Address Mapping Example



The PLB BAR and the mask registers are mapped within the DCR address space of the PCI Express port as described in *Table 20-9*. The POM registers are mapped within the Vendor-Specific Extended Capability (VSEC) structure of the PCI Express configuration space (see *Vendor-Specific Extended Capability (VSEC) Structure* on page 623).

See *VSEC Next Capability Offset, Capability Ver, and EC ID Reg (PECFGn_VSECCAP)* on page 623 and *VSEC Length, Revision, and ID Register (PECFGn_VSECID)* on page 623 for more information.

Table 20-9. Outbound Region Registers

Region	DCR registers		Memory Mapped Registers
	PLB BAR	MASK	POM
Memory #1	PEGPLn_OMR1BAH PEGPLn_OMR1BAL	PEGPLn_OMR1MSKH PEGPLn_OMR1MSKL	PECFGn_POM0LAH PECFGn_POM0LAL
Memory #2	PEGPLn_OMR2BAH PEGPLn_OMR2BAL	PEGPLn_OMR2MSKH PEGPLn_OMR2MSKL	PECFGn_POM1LAH PECFGn_POM1LAL
Memory #3	PEGPLn_OMR3BAH PEGPLn_OMR3BAL	PEGPLn_OMR3MSKH PEGPLn_OMR3MSKL	PECFGn_POM2LAH PECFGn_POM2LAL
I/O			
Message	PEGPLn_MSGBAH PEGPLn_MSGBAL	PEGPLn_MSGMSK	PECFGn_POM3LAH PECFGn_POM3LAL
Configuration	PEGPLn_CFGBAH PEGPLn_CFGBAL	PEGPLn_CFGMSK	PECFGn_POM4LAH PECFGn_POM4LAL
Register	PEGPLn_REGBAH PEGPLn_REGBAL	PEGPLn_REGMSK	NA

20.13.3 Inbound Address Mapping

The PCI Express master interface supports up to four independent address regions in the PCI Express address space.

- Memory region #0 for inbound memory requests. This region may be split into two non concatenated PLB memory regions. The PECFGn_PIM01SAL and PECFGn_PIM01SAH registers determine the offset between these two regions within the PCI Express address space.
- I/O region.
- Memory region #1 for inbound memory requests. This region may be split into two non concatenated PLB memory regions. The PECFGn_PIM34SAL and PECFGn_PIM34SAH registers determine the offset between these two regions within the PCI Express address space.
- Expansion ROM.

The size of each of these four regions is programmable as described in *Table 20-11* on page 515.

Table 20-10. Inbound Memory Ranges

Region	PCI Address	PLB Address	Size (Bytes)	Address Boundary	Access	Comments
Memory #0	BAR0 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	PIM0 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{10} (1KB) to 2^{63}	N × Size	R/W	Transactions are decoded only if they are of memory type.
		PIM1 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{20} (1MB) to 2^{63}			
I/O	BAR1 0000_0000_0000_0000 0000_0000_FFFF_FFFF	PIM2 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{7} (128B) to 2^{15} (32KB)	N × Size	R/W	Transactions are decoded only if they are of I/O type.
Memory #1	BAR2 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	PIM3 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{10} (1KB) to 2^{63}	N × Size	R/W	Transactions are decoded only if they are of memory type.
		PIM4 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{20} (1MB) to 2^{63}			
Expansion ROM	BAR3 0000_0000_0000_0000 0000_0000_FFFF_FFFF	PIM5 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{11} (2KB) to 2^{24} (16MB)	N × Size	R/W	Transactions are decoded only if they are of memory type.

Notes:

1. All address regions are programmable in a location within the PCI Express address space at a granularity of their size. For example, if the size is 4GB, then it begins on a 4-GB boundary.
2. Each region can be disabled or enabled using the corresponding InRegion Valid field defined in the PECFGn_PIMEN register of the Vendor-Specific Extended Capability (VSEC) structure.
3. Memory Region #0 and Memory Region #1 are independently prefetchable if PECFGn_BAR0L[PEF], and respectively, PECFGn_BAR2L[PEF]/PECFGn_RTBAR2L[PFECH] are set.
4. No overlap is allowed between inbound address regions.

Typically, the configuration of these regions is only changed during the initialization process, not during chip operation. The registers that define those regions must only be changed during reset, or after reset *and* when none of the regions is enabled.

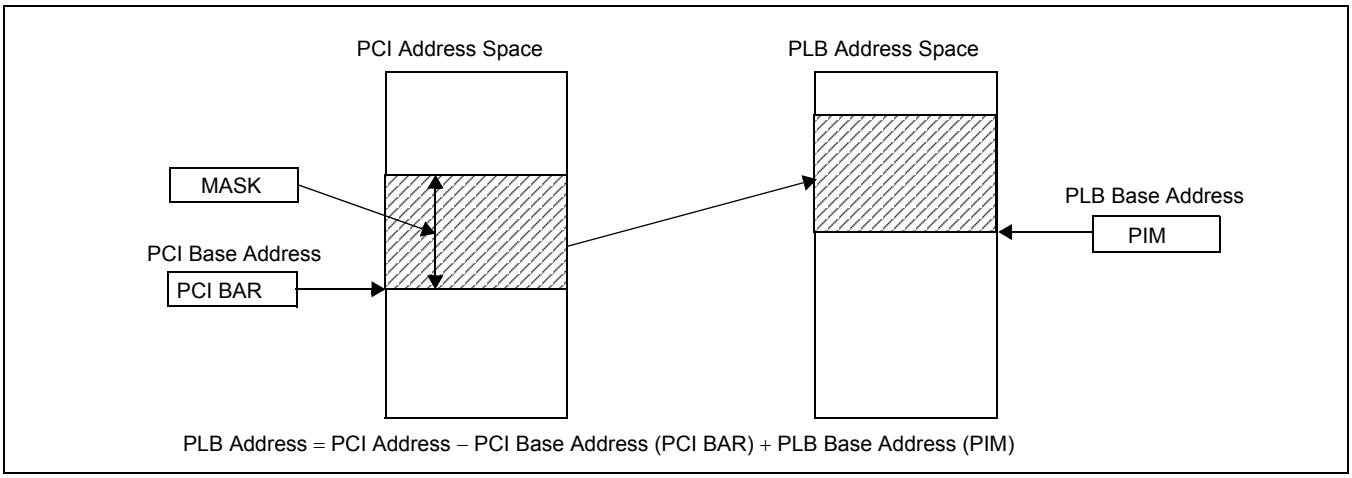
20.13.4 Inbound Address Translation

When an inbound transaction arrives, the PLB Master of the PCI Express port compares the high MSBs of the incoming PCI Express address to each of the four PCI BAR vectors. Only the bits that are masked (set to 1 in the mask vector) are compared against the incoming PCI Express address. If a match occurs in one of the four regions, address translation is applied to the incoming PCI Express address according to the selected region rules.

For address translation, each address region has at least one associated PLB Base Address vector, known as the PCI Inbound Map (PIM). PCI Express-to-PLB address translation is performed by concatenating MSBs of the PIM vector to the LSBs of the incoming PCI Express address. The number of MSBs replaced is defined by the mask vector. Only the MSBs that are masked (set in the mask vector) are taken from the PIM vector. The remaining PCI Express address bits (lsbs) are unchanged. Using the PIM vector, the PLB Master interface claims a region within the PLB address space.

Figure 20-22 shows the Inbound Address Translation logic, using a single PIM. The PCI BAR vector defines a PCI Express address space offset. The PIM vector defines a PLB address space offset. The size of both regions is identical and is determined by the mask vector. The address translation function replaces the PCI Express offset with the PLB space offset by replacing the MSBs from the PCI BAR with MSBs from the PIM vector, leaving all LSBs unchanged.

Figure 20-22. Inbound Address Translation with a Single PIM



For example, assume that the size of the memory region is 1 GB (2^{30}). The mask vector is set so that bits 0:34 are ones and bits 35:63 are zeroes. The offset address of the region is set by PCI Express address bits 0:34, which should be equal to one of the four PCI BARs. Thus, the PLB address is a concatenation of bits 0:34 from the PIM input to bits 35:63 from the PCI Express address, as shown in *Figure 20-23*.

Figure 20-23. Inbound Address Mapping Example

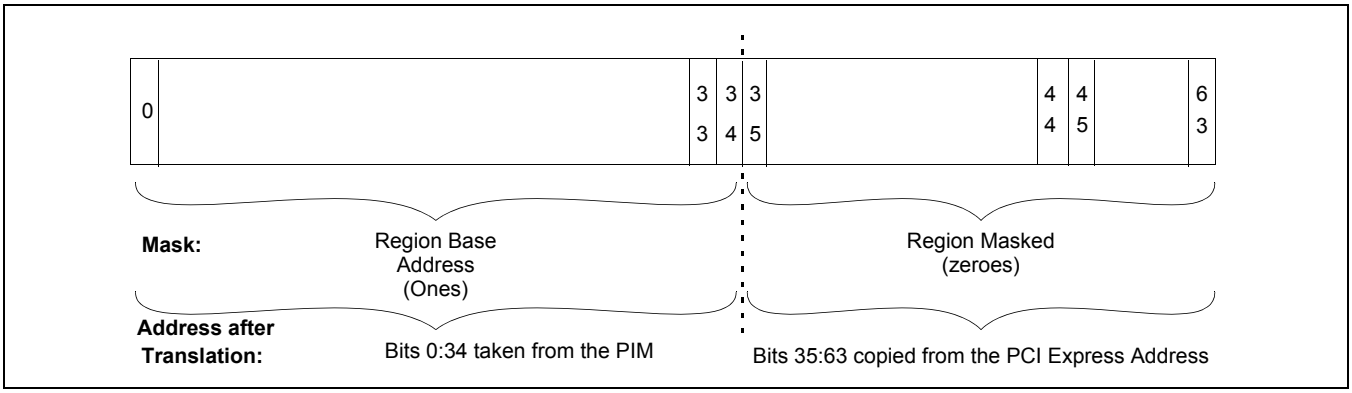
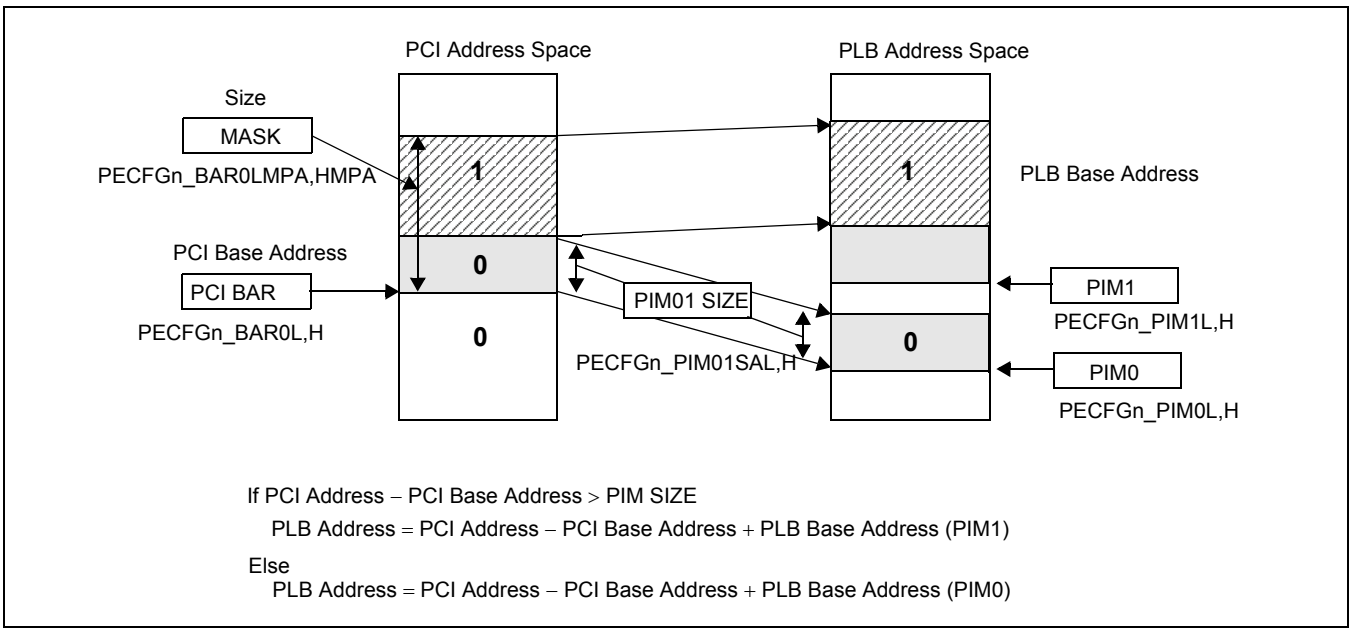


Figure 20-24 shows the Inbound Address Translation logic, using a double PIM. This logic only applies to Memory Region #0 and Memory Region #1. Before applying the address translation function described previously for a single PIM, the PCI address is compared to an offset (PIM SIZE) that splits the address region into two regions.

Transactions located below the offset are translated into the PLB address space using the first PIM (PIM0). All other transactions are translated using the second PIM (PIM1). The value of the offset (PIM SIZE) is programmable and can range from 1 KB to the size of the address region defined by the mask.

Figure 20-24. Inbound Address Translation with a Double PIM



The PCI BAR, the mask, the PIM, and the PIM SIZE registers are mapped within the PLB configuration address space of the PCI Express port, as shown in *Figure 20-11*.

Table 20-11. Inbound Region Registers

Address Space	PLB Space			
Region	**PCI BAR**	**MASK**	**PIM**	**PIM SIZE**
Memory #0	PECFGn_BAR0H PECFGn_BAR0L	PECFGn_BAR0HMPA PECFGn_BAR0LMPA	PECFGn_PIM0LAH PECFGn_PIM0LAL PECFGn_PIM1LAH PECFGn_PIM1LAL	PECFGn_PIM01SAH PECFGn_PIM01SAL
I/O	**Endpoint** PECFGn_BAR1 **Root Port** PECFGn_RTBAR1	PECFGn_BAR1MPA	PECFGn_PIM2LAH PECFGn_PIM2LAL	NA
Memory #1	**Endpoint** PECFGn_BAR2H PECFGn_BAR2L **Root Port** PECFGn_RTBAR2H PECFGn_RTBAR2L	PECFGn_BAR2LMPA PECFGn_BAR2HMPA	PECFGn_PIM3LAH PECFGn_PIM3LAL PECFGn_PIM4LAH PECFGn_PIM4LAL	PECFGn_PIM34SAH PECFGn_PIM34SAL
Expansion ROM	**Endpoint** PECFGn_EROMBA **Root Port** PECFGn_RTEROM	**Endpoint** PECFGn_EROMBAPA **Root Port** PECFGn_RTEROMPA	PECFGn_PIM5LAH PECFGn_PIM5LAL	NA

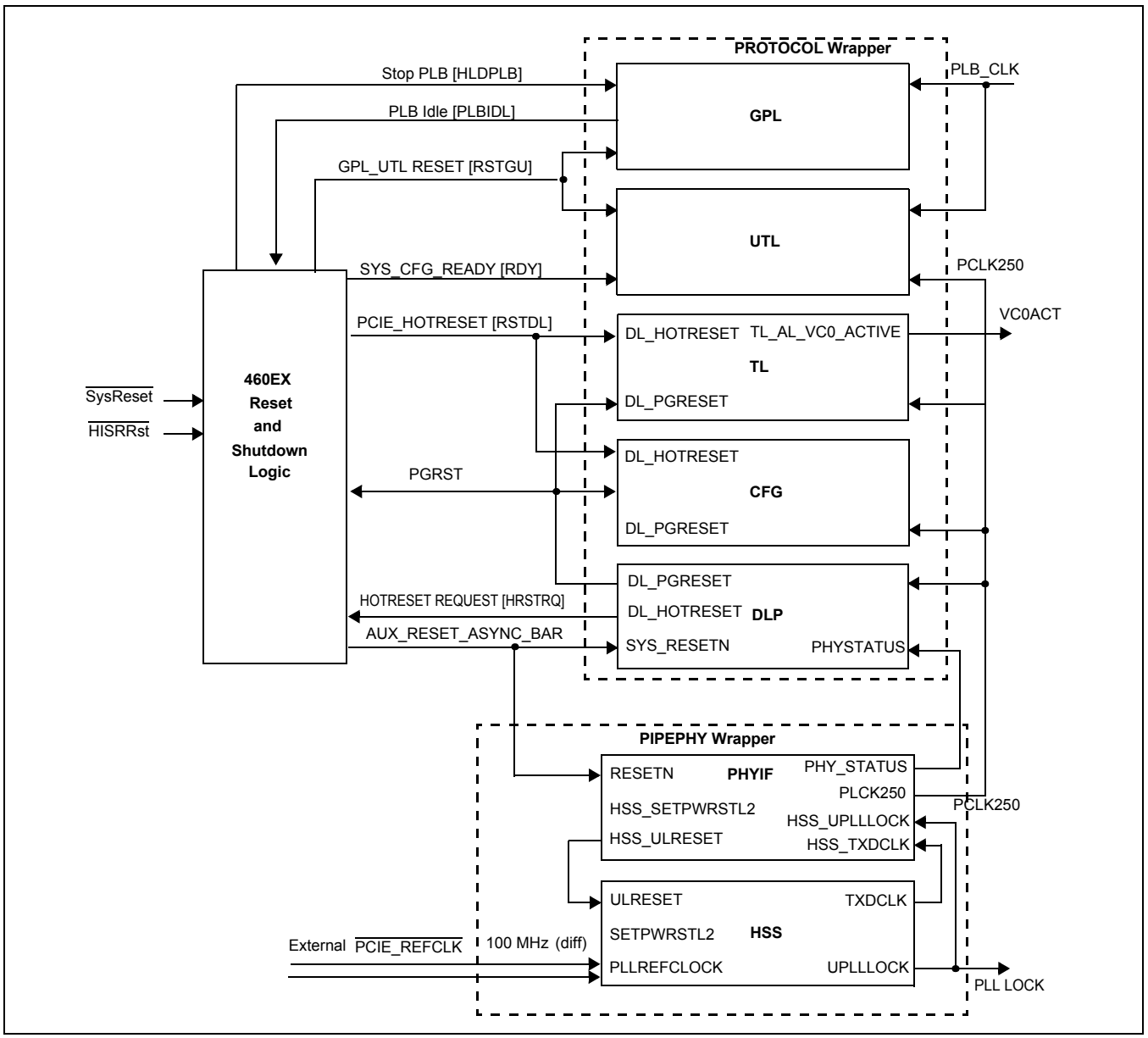
20.14 PCI Express Port Reset Sequences

The *PCI Express Base Specification* defines the following primary reset sequences:

- Hardware Reset, also called Cold Reset
- Software Reset, also called Warm Reset
- Reset initiated by the Root Port, also called Hot Reset
- Port Shutdown

Figure 20-25 shows the relationship between some significant fields in specific SDR registers that are dedicated to the PCI Express interfaces and the respective PCI Express port stack.

Figure 20-25. PCI Express Port Stack and Reset



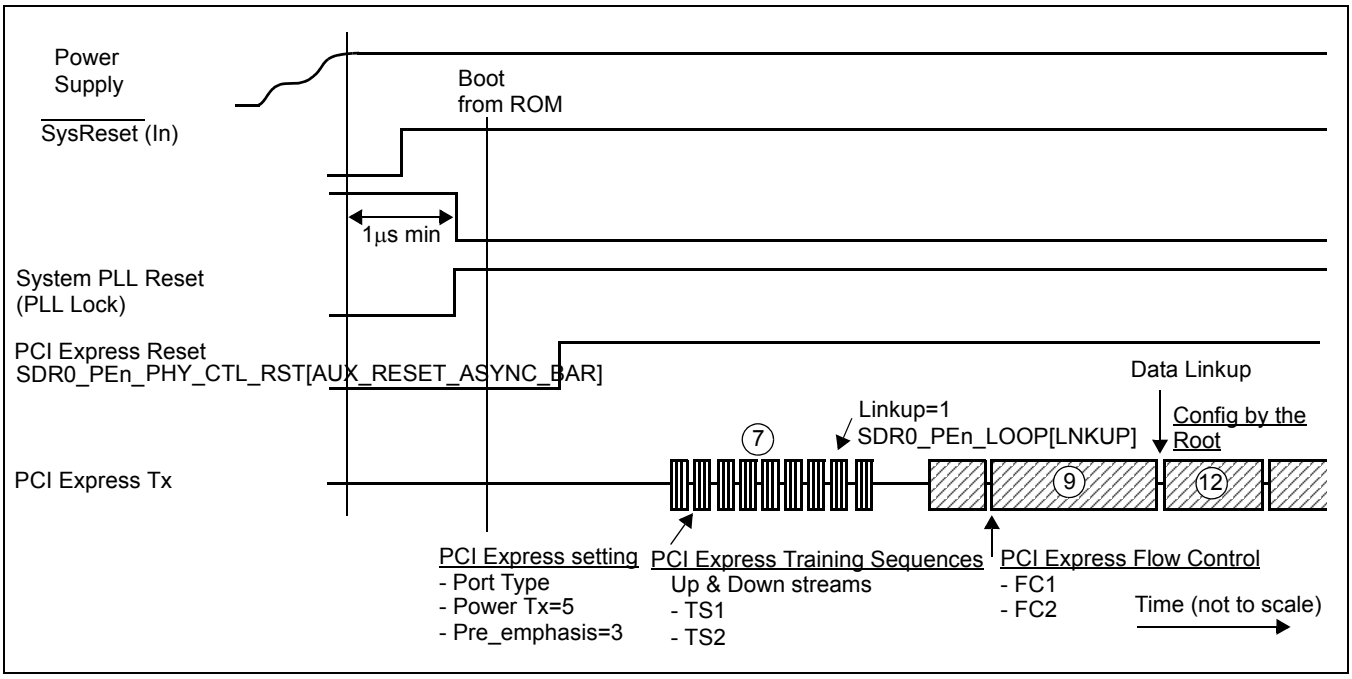
20.14.1 Initialization After System Reset (SysReset): Cold Reset

While SysReset inputs are asserted, all PCI Express ports are in reset mode. PCI Express output drivers are in high impedance state. In reset mode, bits in the following PCI Express-related SDR registers are set as indicated:

- RSTGU SDR0_PEn_RCSSET = 1 GPL and UTL in Reset
- RSTDL SDR0_PEn_RCSSET = 1 CFG and TL in Reset
- AUX_RESET_ASYNC_BAR SDR0_PEn_PHY_CTL_RST = 0 PHY in Reset (Active low signal)
- HLDPLB SDR0_PEn_RCSSET = 0
- RDY SDR0_PEn_RCSSET = 0

Figure 20-26 shows the timing relationship of events that occur for PCI Express initialization in the 460EX/EXr/GT.

Figure 20-26. PCI Express Initialization Timing Sequence



The following sequence is recommended for performing a hardware reset of the PCI Express port:

1. Primary steps of the PPC460EX/EXr/GT overall initialization:
   a. De-assert SysReset.
   b. Set chip default values with or without external serial Bootstrap ROM.
   c. Lock System PLL.
2. Download code from the Boot ROM with specific PCI Express settings as follows:
   a. Port Type: SDR0_PEn_DLPSET[PTYPE] = Root/End Point
   b. Driver level: SDR0_PEn_LnDRV[DRV_LVL] =
   c. Swing: SDR0_PEn_LnDRV[EMP_POST] =
3. Software sets SDR0_PEn_PHY_CTL_RST[AUX_RESET_ASYNC_BAR] to 1 (active low) and SDR0_PEn_RCSSET[RSTGU] to 0 (during the same DCR write operation), which de-asserts the AUX_RESET_ASYNC_BAR and RSTGU signals.
   a. De-asserting the AUX_RESET_ASYNC_BAR signal enables the High Speed Serial (HSS) Link PLL.
   b. SDR0_PEn_PHYSTA[CMULOCK] from the SerDes—Provides the status of the locking mechanism, but does not trigger any software action.
   c. HSS output drivers are enabled automatically.

4. The PIPEPHY wrapper notifies the PROTOCOL wrapper when its initialization is complete.
   a. The DLP function sets SDR0_PEn_RCSSTS[PGRST] to 0 (which de-asserts the PGRST signal), to indicate that the PCI Express clocks are stable.
5. The PCI Express port can start Link Training Sequences (TS1—TS2) on the PCI Express interface when the clocks are stable. It is recommended to wait for 100μs to allow HSS Link training to complete.
6. During this time, software polls SDR0_PEn_RCSTS[PGRST]. When this bit changes to 0, software performs the following PCI Express port configuration actions:
   a. Configures the PCI Express internal registers (including the UTL buffers, by means of DCR operations)
   b. Configures Root Port and Endpoint Configuration space
   c. Sets PEUTLn_PCTL[VRB] to 1, to enable Flow Control initialization
7. When both Step 5 and Step 6 are completed, the PCI Express port performs the Flow Control initialization sequence.
8. When Flow Control initialization is complete, the PCI Express port sets SDR0_PEn_RCSSTS[VC0ACT] to 1, which asserts the VC0ACT signal for both Root Port and Endpoint.
9. When the PCI Express port is fully initialized and able to process configuration transactions, software sets SDR0_PEn_RCSSET[RDY] to 1, which asserts the RDY signal.
10. Configuration by the Root Port: (Address PIM, POM, MSI, IRQ):
    a. Root Port:
       (1) The Root Port software performs Endpoint initialization by processing PCI Express configuration space transactions. The last write operation sets PECFGn_CMDSTATUS[PME] to indicate bus master enable.
    b. Endpoint:
       (1) The Endpoint software polls SDR0_PEn_RCSSTS[BMEN]. This bit is a copy of PECFGn_CMDSTATUS[PME].
       (2) When SDR0_PEn_RCSSTS[BMEN] changes to 1, PCI Express Endpoint initialization is complete.
11. After both the Root Port and the Endpoint are initialized, the state of the PCI Express link is monitored at both ends.
    a. The BMEN signal and the VC0ACT signal are ANDed to generate the BUSMASTER_VC0 signal.
    b. When the link goes down, the BUSMASTER_VC0 signal is de-asserted, which causes UIC2_SR[PE*n*-BMV] to be set to 0 (where *n* = the PCI Express port number).

Note: While RDY is de-asserted, the UTL function permits read/write access to configuration space from the PLB. After the VC0ACT signal is asserted, the UTL function (as an Endpoint) responds to configuration requests with configuration Retry Status (CRS).

20.14.2 Reset Initiated by Software: Warm Reset

The following sequence is recommended for performing a software reset of the PCI Express port:

1. Software sets SDR0_PEn_RCSSET[HLDPLB] to 1:
   a. PCI Express PLB Master Interface:
      (1) Active inbound transactions on the PLB bus are completed. Pending inbound transactions are aborted.
      (2) Because the PLB Master is no longer processing inbound transactions, PCI Express internal buffers may become full. To avoid buffer overflow, Flow Control stops any inbound transactions.
   b. PCI Express PLB Slave interface completes current transactions, then rearbitrates any new request.
2. Software polls SDR0_PEn_RCSSTS[PLBIDL]. When both the Master and Slave PLB interfaces have completed the current transactions, the PCI Express port sets this bit to 1 and asserts the PLBIDL signal.

3. Software sets SDR0_PEn_PHY_CTL_RST[AUX_RESET_ASYNC_BAR] to 0 and SDR0_PEn_RCSSET[RSTGU] to 1 (during the same DCR write operation), which asserts the AUX_RESET_ASYNC_BAR signal (active low) and the RSTGU signal.
   a. Asserting the AUX_RESET_ASYNC_BAR signal automatically disables the PCI Express port output drivers.

20.14.3 Exit From a Reset Initiated by Software: Warm Reset Exit

The following sequence is recommended to exit from a reset initiated by the software:

1. Software sets SDR0_PEn_RCSSET[HLDPLB] to 0.
2. Same as Step 3 through Step 10 described in the initialization sequence at *Initialization After System Reset (SysReset): Cold Reset* on page 517.

20.14.4 Reset Initiated by Root Port: Hot Reset

The following sequence is recommended for performing a software reset of the far-end PCI Express port:

1. Root Port side:
   a. Software sets PECFGn_BCR[SBR] to 1, which triggers a Hot Reset on the corresponding PCI Express port.
   b. The PCI Express Training Control Reset bit of the TS1 and TS2 Ordered Sets causes automatic propagation of this reset to the Endpoint.
2. Endpoint side:
   a. The reset assertion from the Root Port is detected and the HRSTRQ signal is asserted.
   b. Assertion of the HRSTRQ signal causes UIC2_SR[PE*n*HRR] to be set to 1 (*n* indicating the PCI Express port number).
   c. Software sets SDR0_PEn_RCSSET[HLDPLB] to 1:
      (1) PCI Express PLB Master Interface:
         - Active inbound transactions on the PLB bus are completed. Pending inbound transactions are aborted.
         - Because the PLB Master is no longer processing inbound transactions, PCI Express internal buffers may become full. To avoid buffer overflow, Flow Control stops any inbound transactions.

      (2) PCI Express PLB Slave interface completes current transactions, then rearbitrates any new request.
   d. Software polls SDR0_PEn_RCSSTS[PLBIDL]. The PCI Express port sets this bit to 1 and asserts the PLBIDL signal when both the Master and Slave PLB interfaces have completed the current transactions.
   e. Software sets both SDR0_PEn_RCSSET[RSTDL] and SDR0_PEn_RCSSET[RSTGU] to 1, which asserts the RSTDL and RSTGU signals.
      (1) PCI Express upper layers (CFG, TL, UTL and GPL) are now in reset mode. However, the link is still active. The Endpoint is now able to receive the Hot Reset de-assertion request from the Root Port.

Note: Execution of Step 2.c through Step 2.e must be performed within 2 μs so as not to exceed the maximum Hot Reset assertion time on the link.

Note: Execution of Step 2.c.2 can require up to 130 PLB clock cycles.

20.14.5 Exit From a Reset Initiated by Root Port: Hot Reset Exit

The following sequence is recommended for exiting from a software reset initiated by the Root Port:

1. Root Port side:
   a. Software sets PECFGn_BCR[SBR] to 0, which de-asserts the associated signal.

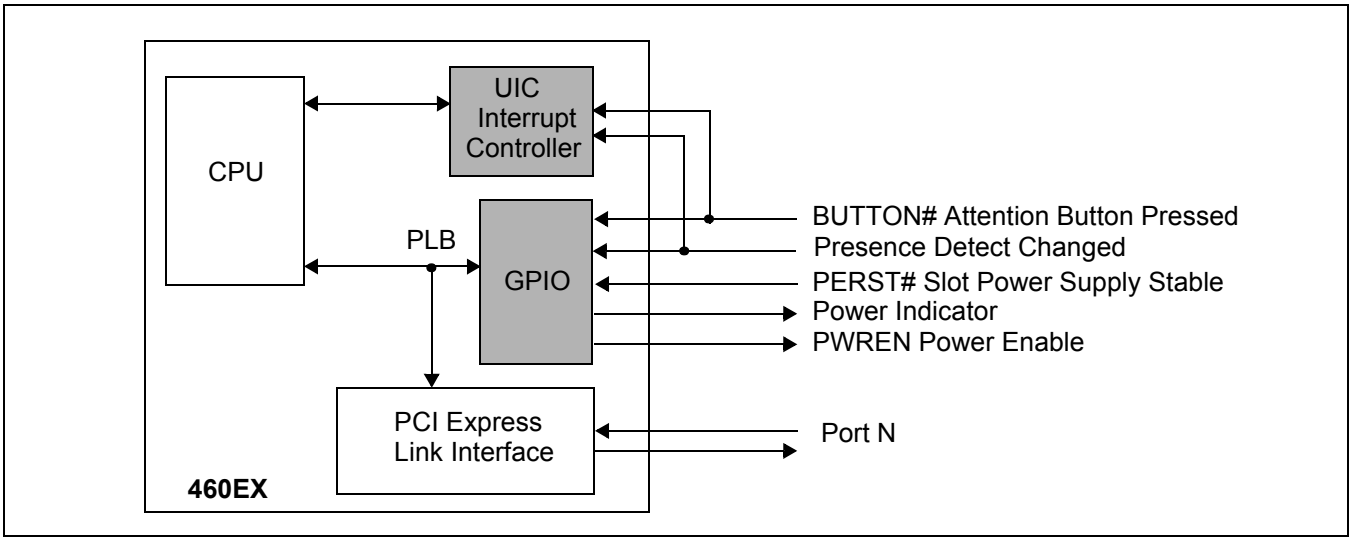
b. This reset de-assertion is propagated automatically to the Endpoint by means of the PCI Express Training Control Reset bit of the TS1 and TS2 Ordered Sets.

2. Endpoint side:
   a. The reset de-assertion is detected and the HRSTRQ signal is de-asserted.
   b. De-assertion of the HRSTRQ signal causes UIC2_SR[PE*n*HRR] to be set to 0 (*n* indicating the PCI Express port number).
   c. Software sets SDR0_PEn_RCSSET[HLDPLB] to 0, which de-asserts the HLDPLB signal.
   d. Software sets SDR0_PEn_RCSSET[RSTGU] to 0, which de-asserts the RSTGU signal for both the GPL and UTL functions.
   e. Same as Step 5 through Step 10 in the initialization sequence described at *Initialization After System Reset (SysReset): Cold Reset* on page 517.

20.15 Hot-Plug Capability

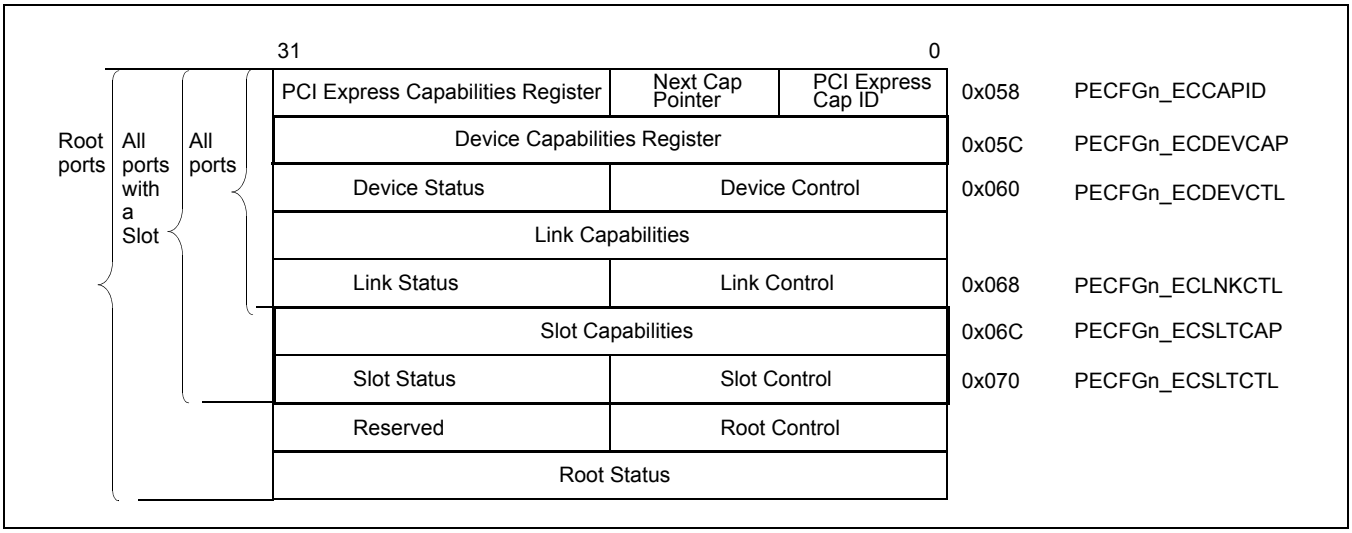
The PCI Express ports do not provide native Hardware/Software Hot-Plug support. However, existing software and hardware can be used to exploit some Hot-Plug capability. Some GPIOs and external interrupts can be dedicated to Hot-Plug capability. Two external interrupts are dedicated to the Presence Detect Changed and Attention Button Pressed signals. One GPIO input is provided for the Slot Power Supply Stable (PERST#) signal, and two GPIO outputs are provided for the Power Indicator and Power Enable signals as shown in *Figure 20-27*.

Figure 20-27. Hot Plug Capability



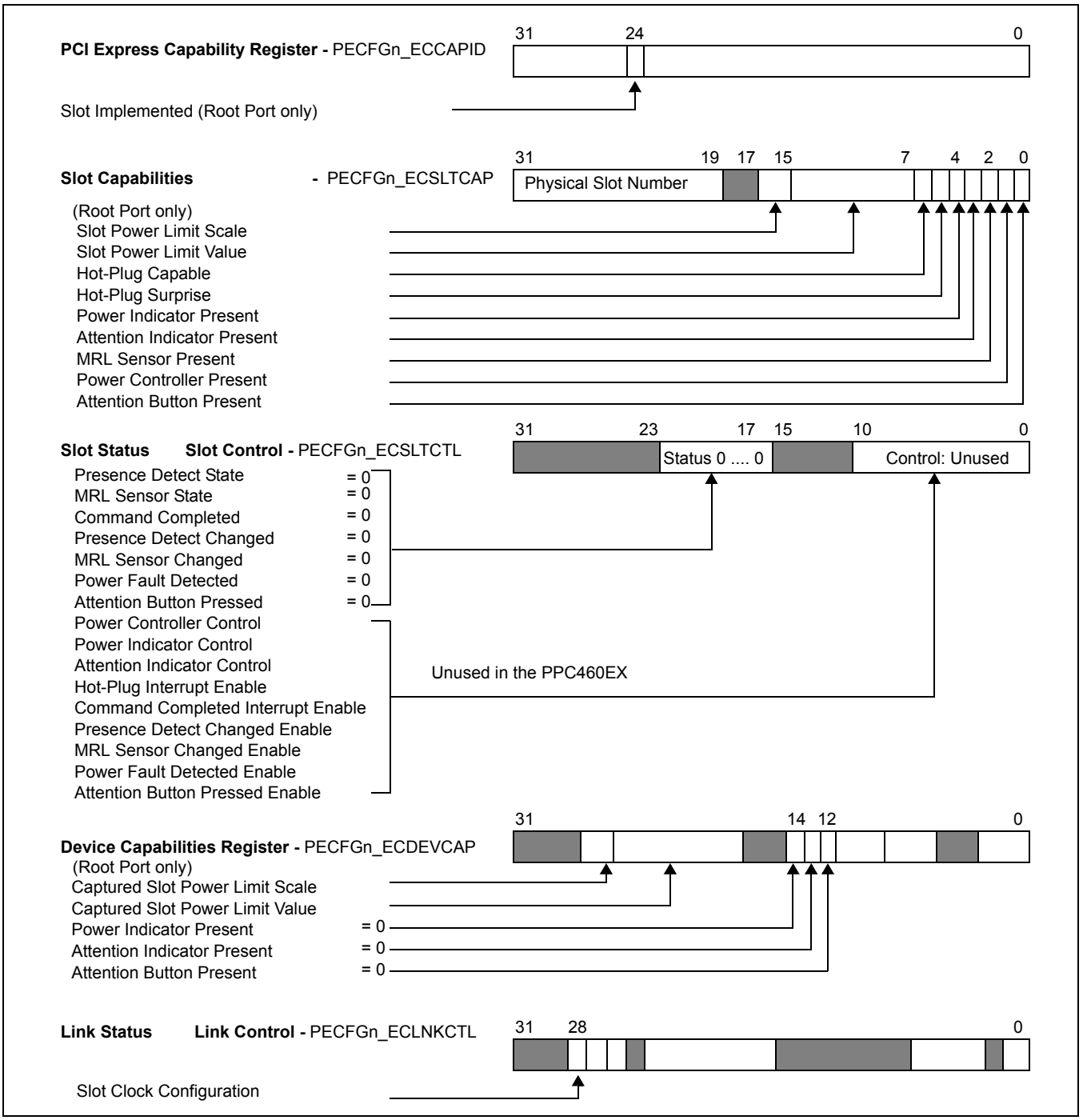
The PCI Express configuration space includes registers to deal exclusively with Hot-Plug capability. The following sections describe the Hot-Plug capability registers defined in the PCI Express standard, depending on the type of the port (Root Port, Endpoint, or both), as shown in *Figure 20-28*.

Figure 20-28. Hot-Plug Capability Register Set



The PCI Express port configuration space includes Hot-Plug capability information as shown in *Figure 20-29*.

Figure 20-29. Hot-Plug Capabilities in the PCI Express Port Configuration Space



20.16 Vital Product Data

Vital product Data (VPD) can contain unique information about the hardware and software elements of a system. The VPD can include information about Field Replaceable Units (FRUs) such as part number and serial number, as well as performance and failure data for the device being monitored. This is done by reading the information from the hardware, software, and microcode components.

The VPD resides in a storage device (for example, in serial EEPROM in a PCI device). Access to the VPD is provided by means of registers in the PCI Express VPD Capability Structure, which is contained in the PCI Configuration Space structure. See *PCI Express VPD Capability Structure* on page 614 for more information and a description of the VPD registers.

20.16.1 VPD Read Access

The following sequence is recommended for a Root Port (RP) read access of the Vital Product Data (VPD) information located in the Endpoint (EP):

1. Root Port side:
   a. Software writes to the Endpoint VPD Capability ID Register (PECFGn_VPDCAP[FLAG] and [ADDR] to set the FLAG bit to 0 and to specify the address to access within the Endpoint memory space.
   b. Software polls the Endpoint PECFGn_VPDCAP[FLAG] until the Endpoint software sets it to 1 (see Step 2f).
2. Endpoint side:
   a. The Root Port write access to PECFGn_VPDCAP is detected.
   b. A VPD interrupt is generated, which causes UIC2_SR[PE*n*VPD] to be set to 1 (*n* indicating the PCI Express port number).
   c. Software reads PECFGn_VPDCAP[ADDR].
   d. Software reads the data at the address provided by the previous read operation and writes it into PECFGn_VPDDATA bits 31:0.
   e. Software sets UIC2_ER[PE*n*VPD] to 0, which disables VPD interrupts (*n* indicating the PCI Express port number). This prevents the write to PECFGn_VPDCAP[FLAG] in Step f from causing an interrupt, because the Endpoint does not know whether the VPD read access is coming from the PLB interface (Endpoint) or the PCI interface (Root Port).
   f. Software sets the Endpoint PECFGn_VPDCAP[FLAG] to 1.
   g. Software sets UIC2_ER[PE*n*VPD] to 1, which re-enables VPD interrupts (*n* indicating the PCI Express port number).
3. Root Port side:
   a. Checks that the Endpoint PECFGn_VPDCAP[FLAG] is set to 1 (see Step 1b).
   b. Software reads Endpoint PECFGn_VPDDATA bits 31:0. This completes the Root Port VPD read access.

20.16.2 VPD Write Access

The following sequence is recommended for a Root Port (RP) write access of the Vital Product Data (VPD) information located in the Endpoint (EP):

1. Root Port side:
   a. Software writes to the Endpoint VPD Data Register (PECFGn_VPDDATA bits 31:0) with the data to be written in the Endpoint memory space.
   b. Software writes to the Endpoint VPD Capability ID Register (PECFGn_VPDCAP[FLAG] and [ADDR] to set the FLAG bit to 1 and to specify the address to access within the Endpoint memory space.
   c. Software polls the Endpoint PECFGn_VPDCAP[FLAG] until Endpoint software sets it to 1 (see Step 2f).

2. Endpoint side:
   a. The Root Port write access to PECFGn_VPDCAP is detected.
   b. A VPD interrupt is generated, which causes UIC2_SR[PE*n*VPD] to be set to 1 (*n* indicating the PCI Express port number).
   c. Software reads PECFGn_VPDDATA.
   d. Software writes the data to the Endpoint memory space at the address specified in PECFGn_VPDCAP[ADDR].
   e. Software sets UIC2_ER[PE*n*VPD] to 0, which disables VPD interrupts (*n* indicating the PCI Express port number). This prevents the write to PECFGn_VPDCAP[FLAG] in Step f from causing an interrupt, because the Endpoint does not know whether the VPD write access is coming from the PLB interface (Endpoint) or the PCI interface (Root Port).
   f. Software sets the Endpoint PECFGn_VPDCAP[FLAG] bit to 1.
   g. Software sets UIC2_ER[PE*n*VPD] to 1, which re-enables VPD interrupts (*n* indicating the PCI Express port number).
3. Root Port side:
   a. Checks that the Endpoint PECFGn_VPDCAP[FLAG] bit is set to 1 (see Step 1c). This completes the Root Port VPD write access.

20.17 Built-In Self Test (BIST) for PCI Express Ports

Each PCI Express port implements a built-in self test (BIST) to check the physical layer of the PCI express. The BIST consists of a pattern generator and pattern checker. The pattern generator is used to transmit a variety of patterns. The pattern checker locks onto the incoming data, recognizes the pattern being sent, and checks for errors in the received parallel words. The pattern checker is used to check data from the local pattern generator through on-chip or off-chip serial loopback or from a remote transmitter.

The following table lists the data patterns supported by the BIST. The pattern is selected by PEx_BIST_MODE for both the pattern checker as well as the pattern generator. The pattern generated is the raw pattern shown in the table (there are no framing bits, headers, or delimiters sent along with the data). The data pattern repeats continuously as long as it is selected.

Table 20-12. BIST Modes of Operation

SDR0_PEn_BIST[MODE]	Test Pattern [1]
0000	Normal data
0001	K28.5 +/– (0011111010 1100000101 ...)
0010	D21.5 (1010101010 ...)
0011	K28.7 (0011111000 ...)
0100	PCI Express compliance pattern (00111110101010101010110000010 1 0101010101 ...)
0101	D24.3 +/– (1100110011 0011001100 ...)
0110	All 0s
0111	All 1s
1000	PRBS7 (2^7-1)
1001	PRBS10 ($2^{10}-1$)
1010	PRBS15 ($2^{15}-1$)
1011	PRBS23 ($2^{23}-1$)
1100	PRBS31 ($2^{31}-1$)
1101	Reserved
1110	Reserved
1111	Reserved
Note 1: Bit streams are ordered from left (transmitted/received first) to right (transmitted/received last).	

There are three types of patterns:
- 8B10B test patterns (0001, 0010, 0011, 0100, and 0101)
- General test patterns (0110 and 0111)
- PRBS patterns (1000,1001,1010,1011)

Note: The PRBS patterns are not valid 8B10B patterns and have longer bit run lengths.

The 8B10B patterns include the three standard 8B10B test patterns. These are a low frequency test pattern (K28.7), a high frequency test pattern (D21.5), and a mixed frequency test pattern (K28.5+/–). A mixed frequency compliance pattern specified for PCI Express is included as well as a 11/00 pattern that can be used to stress the serializer.

The general test patterns consist of an all-1s pattern and an all-0s pattern. These can be used to check DC characteristics of the driver such as launch amplitude.

The BIST also includes four PRBS patterns of various lengths. The polynomials shown are the feedback taps points for a standard linear feedback shift register (LFSR) implementation. The higher the order of the polynomial, the longer run lengths that are part of the pattern produced. Higher order polynomials are more stringent tests of the SerDes. All PRBS patterns repeat at a period related to the order of the polynomial (2^n-1). For example, the PRBS31 pattern will repeat after $2^{31}-1$ bits.

BIST Control

The pattern generator is reset using SDR0_PEx_LnBIST[BIST_Tx_RESET] which is an asynchronous input that resets the BIST when high. The falling edge of SDR0_PEx_LnBIST[BIST_Tx_RESET] is internally synchronized to the BIST transmit clock and pattern generation starts when it goes low.

At the rising edge of SDR0_PEx_LnBIST[BIST_FORCE_ERR], a single bit error is forced in the BIST output stream. SDR0_PEx_LnBIST[BIST_FORCE_ERR] is also an asynchronous input that is internally synchronized to the internal BIST transmit clock. Therefore, there is no control on exactly which bit in the output stream is inverted.

The pattern checker is reset using SDR0_PEx_LnBIST[BIST_Rx_RESET], which is also an asynchronous signal that is internally synchronized to the BIST receive clock. SDR0_PEx_LnBIST[BIST_Rx_RESET] should be pulsed after clock recovery has locked and valid data is being received. There is no need to attempt to synchronize the deassertion of SDR0_PEx_LnBIST[BIST_Rx_RESET] to any data in the incoming data stream. However, valid data of the pattern selected by SDR0_PEx_LnBIST[BIST_MODE] must be available to the pattern checker when SDR0_PEx_LnBIST[BIST_Rx_RESET] is deasserted. Data checking begins immediately upon deassertion of SDR0_PEx_LnBIST[BIST_Rx_RESET].

For 8B10B patterns (0001–0111), the pattern checker first searches for a valid pattern in the incoming stream. Once one valid pattern is found, SDR0_PEx_LnBIST_STATUS[BIST_SYNC] is asserted and errors detection begins. At each parallel word containing one or more errors, SDR0_PEx_LnBIST_STATUS[BIST_ERR_CNT] is incremented and SDR0_PEx_LnBIST_STATUS[BIST_TOG] is inverted. Error checking is stopped and SDR0_PEx_LnBIST_STATUS[BIST_ERR_CNT] and SDR0_PEx_LnBIST_STATUS[BIST_TOG] are frozen when 15 words containing error have been received. SDR0_PEx_LnBIST_STATUS[BIST_ERR] is set on the first error detected. Checking resumes when SDR0_PEx_LnBIST[BIST_BER_CLEAR] or SDR0_PEx_LnBIST[BIST_Rx_RESET] are pulsed resetting SDR0_PEx_LnBIST_STATUS[BIST_ERR_CNT] to 0, and clearing SDR0_PEx_LnBIST_STATUS[BIST_ERR] and SDR0_PEx_LnBIST_STATUS[BIST_TOG]. SDR0_PEx_LnBIST[BIST_BER_CLEAR] clears the error counter without forcing relock of the pattern checker. SDR0_PEx_LnBIST_STATUS[BIST_PATTERN_SYNC] is not used for 8B10B patterns.

For PRBS patterns (1000–1011), the pattern checker internally self-synchronizes to the incoming PRBS pattern at the deassertion of reset. SDR0_PEx_LnBIST_STATUS[BIST_SYNC] is, therefore, continuously high. SDR0_PEx_LnBIST_STATUS[BIST_ERR_CNT], SDR0_PEx_LnBIST_STATUS[BIST_TOG], SDR0_PEx_LnBIST_STATUS[BIST_ERR], SDR0_PEx_LnBIST[BIST_BER_CLEAR], and SDR0_PEx_LnBIST[BIST_Rx_RESET] operated as described above. It is recommended that SDR0_PEx_LnBIST[BIST_BER_CLEAR], be pulsed after deassertion of SDR0_PEx_LnBIST[BIST_Rx_RESET] waiting at least four parallel clock cycles to allow for data flushing of the internal PRBS checking circuitry. For PRBS patterns, SDR0_PEx_LnBIST_STATUS[BIST_PATTERN_SYNC] will be high on the cycle at the start of the PRBS sequence indicating that an entire PRBS sequence of 2^n-1 bits has been received.

Using the BIST

The following sequence is recommended for using the BIST.

1. Drive the reset signal to the PHY by setting SDR0_PEx_PHY_CTL_RST[AUX_RESET_ASYNC_BAR]=0. While the PHY is being actively reset, ensure that the registers driving the bypass logic are as follows:
   - SDR0_PEx_PHY_TEST[TXCOMPLIANCE][n] = 0 (n = lane)
   - SDR0_PEx_PHY_TEST[RXPOLARITY][n] = 0 (n = lane)
   - SDR0_PEx_PHY_TEST[TXDETRXLOOP] = 0
   - SDR0_PEx_PHY_TEST[POWERDOWN] = 0b10 (P1 state)
   - SDR0_PEx_PHY_TEST[TXELECIDLE][n] = 1 (n = lane)
2. Enable the bypass logic by setting SDR0_PEx_PHY_TEST[TEST_EN] = 1. Enable internal loopback by setting SDR0_PEx_DLPSET[LBACK] = 1 and SDR0_PEx_LnLPBK[SER_LPB_EN] = 1.
3. De-assert the PHY reset (SDR0_PEx_PHY_CTL_RST[AUX_RESET_ASYNC_BAR] = 1) and choose PCIE protocol (SDR0_PEx_PHY_CTL_RST[PROTOCOL] = 0b00). The PHY begins to re-initialize, the PLL locks,

and the PHY indicates successful completion of the power up sequence into the P1 state by asserting SDR0_PEx_OBS[PHYSTATUS] and changing SDR0_PEx_OBS[PSTATE] to a value of 10. In addition, PCLK becomes valid upon entering the P1 state.

4. Initiate a transition to the P0 power state by writing a 0b00 to SDR0_PEx_PHY_TEST[POWERDOWN]. The PHY is now ready for BIST operation.
5. For each lane within the link that is targeted for BIST testing, clear the corresponding SDR0_PEx_PHY_TEST[TXELECIDLE][n] = 0 (n = lane). The lane begins transmitting garbage data that has been encoded for each lane being tested.
6. Set SDR0_PEx_LnBIST[BIST_MODE] to desired pattern on both transmitting and receiving SerDes.
7. Pulse SDR0_PEx_LnBIST[BIST_Tx_RESET] on the transmitting SerDes to 1 and then back to 0. The minimum pulse requirement of 32 Tclk cycles must be met (Tclk = 3.2ns at Gen1 rates). The falling edge of this signal triggers the Tx reset operation.
8. Wait for clock recovery to lock on the receiving SerDes. Wait at least 32 Tclk cycles.
9. Pulse SDR0_PEx_LnBIST[BIST_Rx_RESET] on the receiving SerDes to 1 and then back to 0. The minimum pulse requirement of 32 Tclk cycles must be met (Tclk = 3.2ns at Gen1 rates). The falling edge of this signal triggers the Rx reset operation.
10. Allow at least another 32 Tclk cycles for the receiver to sync up with the incoming stream.
11. Clear bit errors by pulsing the SDR0_PEx_LnBIST[BIST_BER_CLEAR] for a minimum of 32 Tclk cycles pulse width.
12. Monitor the BIST status by reading SDR0_PEx_LnBIST_STATUS. The BIST_ERR and BIST_ERR_CNT fields are only valid if BIST_SYNC is set for non-PRBS patterns and BIST_PATTERN_SYNC is set for PRBS patterns.

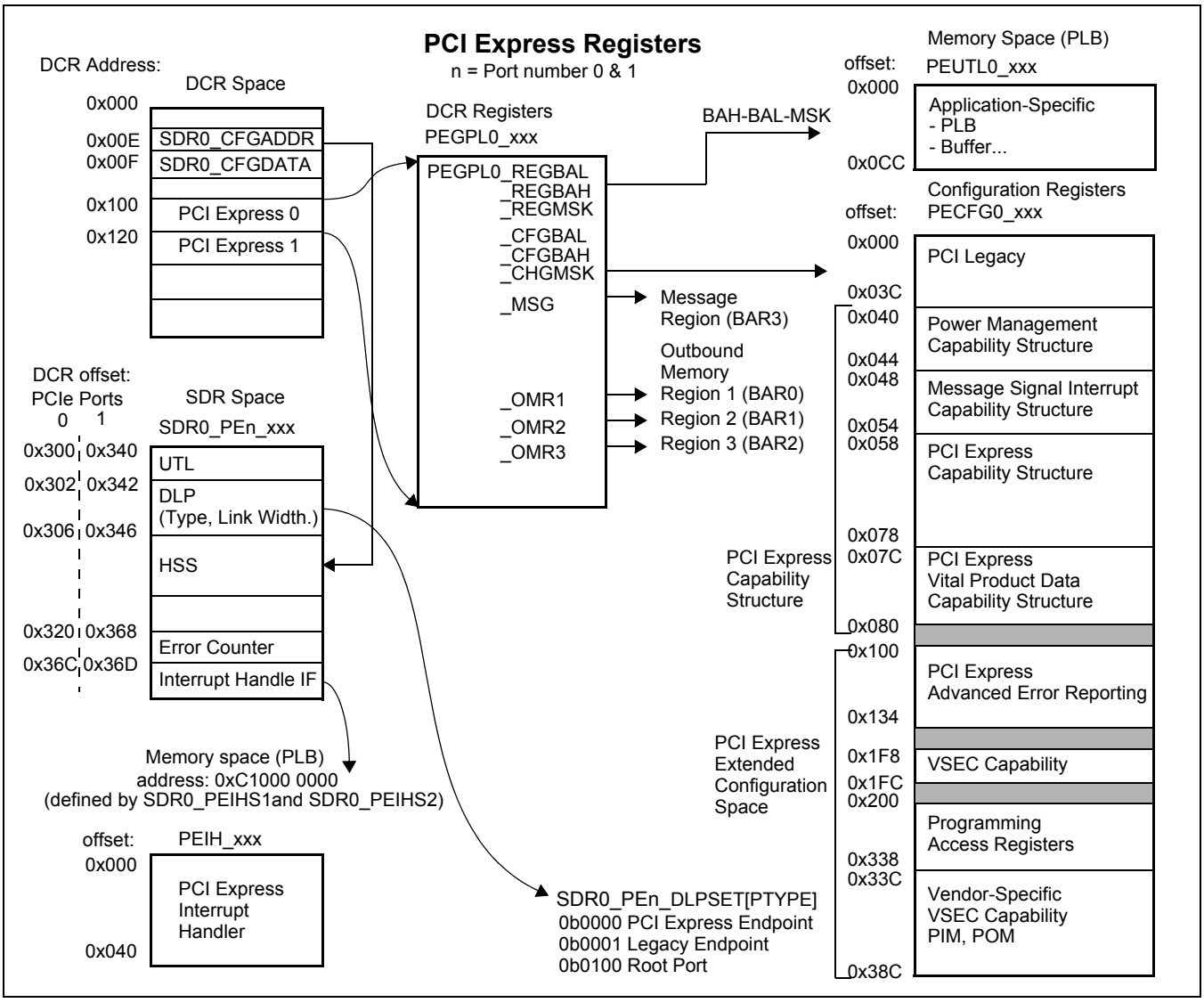
    Note: BIST error indicator bits are valid only when SDR0_PEx_LnBIST_STATUS[BIST_SYNC] is high.
13. BIST errors can be forced by pulsing SDR0_PEx_LnBIST[BIST_FORCE_ERR] = 1 on the transmitting PHY. The rising edge of this signal triggers the insertion of a bit error in the next parallel word to be sent to the serializer. By triggering single bit errors, users can determine that the BIST logic is working by observing the detection of bit errors by reading SDR0_PEx_LnBIST_STATUS register for the receiving PHY.
14. Steps 5–13 can be repeated for testing other lanes, if any.
15. When BIST testing is completed, assert the PHY reset (SDR0_PEx_PHY_CTL_RST[AUX_RESET_ASYNC_BAR]=0). While the PHY reset is asserted, clear SDR0_PEx_PHY_TEST[TEST_EN] = 0 to disable the bypass logic.
16. De-assert PHY reset (SDR0_PEx_PHY_CTL_RST[AUX_RESET_ASYNC_BAR]=1). The PHY must be re-initialize after the testing is completed.

20.18 PCI Express Registers

This section describes the PCI Express registers, which are mapped in several spaces as shown in *Figure 20-30*. These registers are accessible to the programmer (*n* identifies the port number, from 0 to 1):

- SDR0_PEn_xxxx SDR space with indirect access by means of SDR0_CFGADDR and SDR0_CFGDATA. The SDR registers must be programmed before setting any bits in the following registers:
- PEGPLn_xxxx DCR registers primarily responsible for the PLB Slave agent configuration.
- PECFGn_xxxx Memory-mapped configuration register space, organized into:
  - PCI Legacy
  - PCI Express Capability Structure
  - PCI Express Extended Configuration Space
- PEUTLn_xxxx Memory-mapped application-specific space, responsible for the PLB Master agent and PLB bus-to-PCI Express bridge functionality configuration.
- PEIH_xxxx Memory-mapped address defined by SDR0_PEIHS1 and SDR0_PEIHS2 - PCI Express Interrupt Handler.

Figure 20-30. PCI Express Register Mapping



20.19 SDR Configuration Registers for PCI Express Ports (SDR0_PEn_xxxx)

SDR0_PEn_xxxx registers are accessed indirectly through the SDR0_CFGADDR and SDR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions. *Table 3-1* on page 154 lists the DCR addresses for the SDR0_CFGADDR and SDR0_CFGDATA registers.

To read or write one of the SDR0_PEn_xxxx registers, software first writes the register address offset of the target register into the SDR0_CFGADDR register. The target register can then be read or written through the SDR0_CFGDATA DCR register. The following PowerPC code illustrates this procedure by reading the SDR0_PE0_UTLSET1 register:

```
li r3, SDR0_PE0_UTLSET1_Offset
mtdcr SDR0_CFGADDR, r3      ! write SDR0_CFGADDR with the SDR0_UART0 DCR offset address
mfdcr r4, SDR0_CFGDATA      ! read the content of SDR0_PE0_UTLSET1 from SDR0_CFGDATA
```

Table 20-13. PCI Express SDR Registers

Mnemonic	Register	DCR Offset	Access	Page
SDR0_PE0_UTLSET1	PCI Express 0 Upper Transaction Layer Configuration Setting 1	0x0300	R/W	531
SDR0_PE0_UTLSET2	PCI Express 0 Upper Transaction Layer Configuration Setting 2	0x0301	R/W	532
SDR0_PE0_DLPSET	PCI Express 0 Data Link and Logical Physical Configuration	0x0302	R/W	533
SDR0_PE0_LOOP	PCI Express 0 Loopback Interface Status	0x0303	R	534
SDR0_PE0_RCSSET	PCI Express 0 Reset, Clocking, and Shutdown Setting	0x0304	R/W	536
SDR0_PE0_RCSSTS	PCI Express 0 Reset, Clocking, and Shutdown Status	0x0305	R	537
	Reserved	0x0306–0x0307		
SDR0_PE0_L0BIST	PCI Express 0 Lane 0 PHY BIST Control	0x0308	R/W	538
SDR0_PE0_L0BISTST	PCI Express 0 L0 PHY BIST Status	0x0309	R	539
SDR0_PE0_L0CDRCTL	PCI Express 0 L0 CDR Control Register	0x030A	R/W	540
SDR0_PE0_L0DRV	PCI Express 0 L0 Drive Register	0x030B	R/W	542
SDR0_PE0_L0REC	PCI Express 0 L0 Receiver Register	0x030C	R/W	544
SDR0_PE0_L0LPBK	PCI Express 0 L0 Loopback Register	0x030D	R/W	546
SDR0_PE0_L0CLK	PCI Express 0 L0 Clocking Register	0x030E	R/W	548
SDR0_PE0_PHY_CTL_RST	PCI Express 0 PHY Control Reset Register	0x030F	R/W	550
SDR0_PE0_RSTSTA	PCI Express 0 Reset Status Register	0x0310	R	552
SDR0_PE0_OBS	PCI Express 0 Observation Register	0x0311	R	553
SDR0_PE0_PHY_TEST	PCI Express 0 PHY Test Register	0x0312	R/W	554
SDR0_PE0_L0ERRC	PCI Express 0 L0 Error Counter	0x0320	R	555
	Reserved	0x0324–0x033F		
SDR0_PE1_UTLSET1	PCI Express 1 Upper Transaction Layer Configuration Setting 1	0x0340	R/W	531
SDR0_PE1_UTLSET2	PCI Express 1 Upper Transaction Layer Configuration Setting 2	0x0341	R/W	532
SDR0_PE1_DLPSET	PCI Express 1 Data Link and Logical Physical Configuration	0x0342	R/W	533
SDR0_PE1_LOOP	PCI Express 1 Loopback Interface Status	0x0343	R	535
SDR0_PE1_RCSSET	PCI Express 1 Reset, Clocking, and Shutdown Setting	0x0344	R/W	536
SDR0_PE1_RCSSTS	PCI Express 1 Reset, Clocking, and Shutdown Status	0x0345	R	537

Table 20-13. PCI Express SDR Registers (Continued)

Mnemonic	Register	DCR Offset	Access	Page
	Reserved	0x0346–0x0347		
SDR0_PE1_L0BIST	PCI Express 1 Lane 0 BIST Register	0x0348	R/W	539
SDR0_PE1_L1BIST	PCI Express 1 L1 BIST Register	0x0349	R/W	539
SDR0_PE1_L2BIST	PCI Express 1 L2 BIST Register	0x034A	R/W	539
SDR0_PE1_L3BIST	PCI Express 1 L3 BIST Register	0x034B	R/W	539
SDR0_PE1_L0BISTSTS	PCI Express 1 L0 BIST Status Register	0x034C	R	539
SDR0_PE1_L1BISTSTS	PCI Express 1 L1 BIST Status Register	0x034D	R	539
SDR0_PE1_L2BISTSTS	PCI Express 1 L2 BIST Status Register	0x034E	R	539
SDR0_PE1_L3BISTSTS	PCI Express 1 L3 BIST Status Register	0x034F	R	539
SDR0_PE1_L0CDRCTL	PCI Express 1 L0 CDR Control Register	0x0350	R/W	541
SDR0_PE1_L1CDRCTL	PCI Express 1 L1 CDR Control Register	0x0351	R/W	541
SDR0_PE1_L2CDRCTL	PCI Express 1 L2 CDR Control Register	0x0352	R/W	541
SDR0_PE1_L3CDRCTL	PCI Express 1 L3 CDR Control Register	0x0353	R/W	541
SDR0_PE1_L0DRV	PCI Express 1 L0 Drive Register	0x0354	R/W	543
SDR0_PE1_L1DRV	PCI Express 1 L1 Drive Register	0x0355	R/W	543
SDR0_PE1_L2DRV	PCI Express 1 L2 Drive Register	0x0356	R/W	543
SDR0_PE1_L3DRV	PCI Express 1 L3 Drive Register	0x0357	R/W	543
SDR0_PE1_L0REC	PCI Express 1 L0 Receiver Register	0x0358	R/W	545
SDR0_PE1_L1REC	PCI Express 1 L1 Receiver Register	0x0359	R/W	545
SDR0_PE1_L2REC	PCI Express 1 L2 Receiver Register	0x035A	R/W	545
SDR0_PE1_L3REC	PCI Express 1 L3 Receiver Register	0x035B	R/W	545
SDR0_PE1_L0LPBK	PCI Express 1 L0 Loopback Register	0x035C	R/W	547
SDR0_PE1_L1LPBK	PCI Express 1 L1 Loopback Register	0x035D	R/W	547
SDR0_PE1_L2LPBK	PCI Express 1 L2 Loopback Register	0x035E	R/W	547
SDR0_PE1_L3LPBK	PCI Express 1 L3 Loopback Register	0x035F	R/W	547
SDR0_PE1_L0CLK	PCI Express 1 L0 Clocking Register	0x0360	R/W	548
SDR0_PE1_L1CLK	PCI Express 1 L1 Clocking Register	0x0361	R/W	549
SDR0_PE1_L2CLK	PCI Express 1 L2 Clocking Register	0x0362	R/W	549
SDR0_PE1_L3CLK	PCI Express 1 L3 Clocking Register	0x0363	R/W	549
SDR0_PE1_PHY_CTL_RST	PCI Express 1 PHY Control Reset Register	0x0364	R/W	550
SDR0_PE1_RSTSTA	PCI Express 1 Reset Status Register	0x0365	R	552
SDR0_PE1_OBS	PCI Express 1 Observation Register	0x0366	R	553
SDR0_PE1_PHY_TEST	PCI Express 1 PHY Test Register	0x0367	R/W	554
SDR0_PE1_L0ERRC	PCI Express 1 L0 Error Counter	0x0368	R	549
SDR0_PE1_L1ERRC	PCI Express 1 L1 Error Counter	0x0369	R	549
SDR0_PE1_L2ERRC	PCI Express 1 L2 Error Counter	0x036A	R	549
SDR0_PE1_L3ERRC	PCI Express 1 L3 Error Counter	0x036B	R	549
SDR0_PEIHS1	PCI Express Interrupt Handler Interface Setting 1 Register	0x036C	R/W	555

Table 20-13. PCI Express SDR Registers (Continued)

Mnemonic	Register	DCR Offset	Access	Page
SDR0_PEIHS2	PCI Express Interrupt Handler Interface Setting 2 Register	0x036D	R/W	555

20.19.1 PCI Exp. n Upper Transaction Layer Config. Setting 1 (SDR0_PEn_UTLSET1)

Figure 20-31. PCI Express n Upper Transaction Layer Configuration Setting 1 (SDR0_PEn_UTLSET1)

0		Reserved	
1:3	MAXSZ	Maximum Size 000 128 bytes. 001 256 bytes 010 512 bytes	Maximum supported inbound GBIF read request size. Reset value = 0b010 (512 bytes)
4:5		Reserved	
6:7	INTRN	Maximum Inbound Non-posted Transactions 00 4 transactions 01 8 transactions (maximum size should be 256 bytes)	Maximum number of inbound non-posted transactions on system bus (GBIF) Load the value of 2 in PEUTLn_INTR[ICT] if SDR0_PEx_UTLSET2[LKINE] (bit 15) is set to 1. Reset value = 0b00 (4 transactions)
8:9		Reserved	
10:11	INHDN	Maximum Inbound Non-posted Headers 00 2 headers 01 4 headers	Maximum number of inbound non-posted headers that can be stored in the receive buffers. Load in PEUTLn_INHBSZ[INH] if SDR0_PEx_UTLSET2[LKINE] (bit 15) is set to 1. Reset value = 0b00 (2 headers, PE0) Reset value = 0b01 (4 headers, PE1)
12		Reserved	
13:15	IPBFS	Inbound Posted Buffer Size Size of inbound posted data buffer = 0.5 KB All other values are reserved.	Load in PEUTLn_IPDBSZ[IPD] if SDR0_PEx_UTLSET2[LKINE] (bit 15) is set to 1. Reset value = 0b000 (0.5 KB)
16:17		Reserved	
18:19	IPHDN	Maximum Inbound Posted Headers 00 2 headers 01 4 headers	Maximum number of inbound posted headers that can be stored in the receive buffers. Load in PEUTLn_IPHBSZ[IPH] if SDR0_PEx_UTLSET2[LKINE] (bit 15) is set to 1. Reset value = 0b00 (2 headers, PE0) Reset value = 0b01 (4 headers, PE1)
20:21		Reserved	
22:23	ONHDN	Maximum Outbound Non-posted Headers 00 2 headers 01 4 headers	Maximum number of outbound non-posted headers that can be stored in the transmit buffers. Load in PEUTLn_ONHBSZ[ONH]. Reset value = 0b00 (2 headers, PE0) Reset value = 0b01 (4 headers, PE1)
24		Reserved	
25:27	OPBFS	Outbound Posted Buffer Size Size of outbound posted data buffer = 0.5 KB All other values are reserved.	Load in PEUTLn_OPDBSZ[OPD] if SDR0_PEx_UTLSET2[LKINE] (bit 15) is set to 1. Reset value = 0b000 (0.5 KB)
28:29		Reserved	

30:31	OPHDN	Maximum Outbound Posted Headers 00 2 headers 01 4 headers	Maximum number of outbound posted headers that can be stored in the transmit buffers. Load in PEUTLn_OPHBSZ[OPH] Reset value = 0b00 (2 headers, PE0) Reset value = 0b01 (4 headers, PE1)
Note: Please use the default values for all the bits described in this register.			

20.19.2 PCI Exp. n Upper Transaction Layer Config. Setting 2 (SDR0_PEn_UTLSET2)

Figure 20-32. PCI Express x Upper Transaction Layer Configuration Setting 2 (SDR0_PEx_UTLSET2)

0:1		Reserved	
2:3	ONTRN	Maximum Outbound Non-posted Transactions 4 transactions 8 transactions (maximum size should be 256 bytes)	Maximum number of outbound non-posted transactions that can be stored in the transmit buffers. Load the value of 2 in PEUTLn_OUTTR[OCT] if SDR0_PEx_UTLSET2[LKINE] (bit 15) is set to 1. Reset value = 0b00 (4 transactions)
4:6		Reserved	
7	PAREN	Parity Enable	If set to 1, this bit enables odd parity checking and generation on the data buses of the AL/TL interface. This allows single bit error detection, no correction. A packet is nullified when parity error is detected. This field must be stable before setting the AUX_RESET_ASYNC_BAR signal (SDR0_PE0_PHY_CTL_RST[3]) to 1. Reset value = 1 (enable)
8		Reserved	
9:11	ORSQ	Maximum Outbound Read Requests 000 128 bytes 001 256 bytes 010 512 bytes	Maximum supported outbound PCI Express read request. Reset value = 0b010 (512 bytes)
12:14		Reserved	
15	LKINE	Link Init Enable	Buffer allocation registers are initialized to 0 and require software initialization before link activation. Buffer allocation registers are initialized after reset. The allocation register fields affected by this initialization are: PEUTLn_OUTTR[OCT] (Outbound Non Posted Requests) PEUTLn_OPDBSZ[OPD] (Outbound Posted Data Buffer Size) PEUTLn_IPHBSZ[IPH] (Inbound Posted Number of Header Buffers) PEUTLn_IPDBSZ[IPD] (Inbound Posted Data Buffer Size) PEUTLn_INTR[ICT] (Number of Inbound Reads) PEUTLn_INHBSZ[INH] (Inbound Non Posted Number of Header Buffers) Reset value = 1 (enable)
16:31		Reserved	
Please use the default values for all the bits described in this register.			

20.19.3 PCI Express n Data Link and Logical Physical Config. Setting (SDR0_PEn_DLPSET)

Figure 20-33. PCI Express x DLP Configuration Setting Register (SDR0_PEn_DLPSET)

0:2		Reserved	
3	LBACK	Loop Back	If set to 1, the port (both Root Port and Endpoint configurations) enters loopback master mode if at least one lane has detected a receiver when a link is being negotiated. To ensure that loopback mode is entered, it is recommended to assert this signal before setting the AUX_RESET_ASYNC_BAR field (SDR0_PE0_PHY_CTL_RST[3]) to 1. The link remains in loopback mode until the LBACK field is set to 0, at which time it is renegotiated.
4:6		Reserved	
7	SROFF	Scrambling Off	If set to 1, the port disables scrambling on the link during link configuration. This field has no effect once a link is configured. To ensure that scrambling is turned off, it is recommended to set this field to 1 before setting the AUX_RESET_ASYNC_BAR field (SDR0_PE0_PHY_CTL_RST[3]) to 1 during a full stack Endpoint or Root Port Reset Exit sequence or setting the RSTDL field (SDR0_PEx_RCSSET[15]) to 0 during an Endpoint Hot Reset Exit sequence.
8:11	PTYPE	Port Type 0000 PCI Express Endpoint 0001 Legacy PCI Express Endpoint 0100 Root Port All other values are reserved.	This field must be stable before setting the AUX_RESET_ASYNC_BAR field (SDR0_PE0_PHY_CTL_RST[3]) to 1. This field is the programming access of the read only PCI Express Capability Register (PECFG*n*_ECCAPID[23:20]). Reset value = 4b0100 (PE0 - 1 lane, Root Port). Reset value = 4b0001 (PE1 - 4 lanes, Legacy Endpoint).
12:13		Reserved	
14:19	LINKW	Maximum Link Width 000001 (x1) 000100 (x4) All other values are reserved.	This field defines the maximum link width that the PCI Express port attempts to negotiate. This field is the programming access of the read only PCI Express Link Capability register (PECFGn_ECLNKCAP[9:4]). This field must be stable before setting the AUX_RESET_ASYNC_BAR field (SDR0_PE0_PHY_CTL_RST[3]) to 1. Reset value = 6b000001 (PE0 - 1 lane). Reset value = 6b000100 (PE1 - 4 lanes).
20:22		Reserved	
23	ERCEN	Error Counter Enable	If master loopback mode (SDR0_PEx_DLPSET[3]) is set to 1, each lane of the link enters a loopback negotiation process. Each lane that receives this sequence without error sets the respective bit of the LBACT field (SDR0_PEx_LOOP[12:15]). From that time each lane is monitored for errors. These errors are accumulated in the SDR0_PEx_LnERRC lane error counters. This field must be stable before setting the LBACK field (SDR0_PEx_DLPSET[3]) to 1.
24:31		Reserved	

20.19.4 PCI Express 0 Loopback Interface Status (SDR0_PE0_LOOP)

Figure 20-34. PCI Express 0 Loopback Interface Status Register (SDR0_PE0_LOOP)

0:2		Reserved	
3	IBPRE	Inbound Presence 0 1 Asserted when the PCI Express function has detected that a remote component is connected on its port.	
4:14		Reserved	
15	LBACT	Loopback Active	Master Loopback Mode: If master loopback mode (SDR0_PE0_DLPSET[3]) is set to 1, lane 0 of the link enters a loopback negotiation process. If lane 0 receives this sequence without error, then LBACT is set to 0b1. From that time lane 0 is monitored for errors. These errors are accumulated in the SDR0_PE0_L0ERRC counter. If lane 0 receives the loopback negotiation sequence with error(s), then LBACT is not set. The link returns to the detect state after a 2ms timeout.
16:18		Reserved	
19	LNKUP	Link Up 0 Currently no link is established with a remote port. In this state, the TL function is automatically reset 1 This field indicates that the DLP function has performed the link training sequence and brought the link to the Link Up state. It is now up to the TL function to initialize flow control and request initial credits update before attempting to transmit any TLPs or DLLPs	If the LNKUP field is set to 0 during the flow control initialization sequence, this initialization is considered cancelled. If the link goes down, all TLPs that were accepted for transmission but not acknowledged by the remote component are considered lost (all TLPs in the replay buffer).
20:22		Reserved	
23	PML1	1 Link in the L1 State 0 1 Link in the L1 state (whether because of an active state L1 request or a standard L1 request). The transaction layer must not attempt to transmit a TLP if this field is set to 1	
24:26		Reserved	
27	PML23	1 Link in the L2/L3 Ready State 0 1 Link in the L2/L3 ready state and the PHYIF function has been directed to its P2 power state (HSS powered down)	The transaction layer must not attempt to transmit a TLP if this field is set to 1. **Note:** PML23 field assertion does not guarantee that the PHYIF function is ready for removal of the PCLK250 clock. This event is determined by the 250OFF field in SDR0_PEx_RCSSTS (bit 27).
28:30		Reserved	
31	LKRVL	Link Lanes Reversed 0 Link negotiated with the preferred lane mapping 1 Link negotiated with the reversed lane mapping	x1 link can only be formed on Lane 0 x4 link with lane reversal will map Lane 0 onto lane 3

20.19.5 PCI Express 1 Loopback Interface Status Register (SDR0_PE1_LOOP)

Figure 20-35. PCI Express 1 Loopback Interface Status Register (SDR0_PE1_LOOP)

0:2		Reserved	
3	IBPRE	Inbound Presence 0 1 Asserted when the PCI Express function has detected that a remote component is connected on its port	
4:11		Reserved	
12:15	LBACT	Loopback Active	Master Loopback Mode: If master loopback mode (SDR0_PE1_DLPSET[3]) is set to 1, each lane of the link enters a loopback negotiation process. Each lane that receives this sequence without error sets the respective bit of the LBACT field to 0b1. From that time each lane is monitored for errors. These errors are accumulated in the SDR0_PE1_LnERRC counters. Each lane that receives the loopback negotiation sequence with error(s) does not set the respective bit of the LBACT field. If no bit of the LBACT field is set, this indicates that no lane has successfully completed loopback negotiation. The link returns to the detect state after a 2-ms timeout.
16:18		Reserved	
19	LNKUP	Link Up 0 Currently no link is established with a remote port. In this state, the TL function is automatically reset 1 Indicates that the DLP function has performed the link training sequence and brought the link to the Link Up state. It is now up to the TL function to initialize flow control and request initial credits update before attempting to transmit any TLPs or DLLPs	If the LNKUP field is set to 0 during the flow control initialization sequence, this initialization is considered cancelled. If the link goes down, all TLPs that were accepted for transmission but not acknowledged by the remote component are considered lost (all TLPs in the replay buffer).
20:22		Reserved	
23	PML1	1 Link in the L1 state 0 1 Link in the L1 state (whether because of an active state L1 request or a standard L1 request)	The transaction layer must not attempt to transmit a TLP if this field is set to 1.
24:26		Reserved	
27	PML23	1 Link in the L2/L3 Ready state 0 1 Link in the L2/L3 ready state and the PHYIF function has been directed to its P2 power state (HSS powered down)	The transaction layer must not attempt to transmit a TLP if this field is set to 1. **Note:** PML23 field assertion does not guarantee that the PHYIF function is ready for removal of the PCLK250 clock. This event is determined by the 250OFF field in SDR0_PEx_RCSSTS (bit 27).
28:30		Reserved	
31	LKRVL	Link Lanes Reversed 0 Link negotiated with the preferred lane mapping 1 Link negotiated with the reversed lane mapping	x1 link can only be formed on Lane 0 x4 link with lane reversal will map Lane 0 onto lane 3

20.19.6 PCI Express n Reset, Clocking, and Shutdown Setting (SDR0_PEn_RCSSET)

Figure 20-36. PCI Express n Reset, Clocking and Shutdown Setting Register (SDR0_PEn_RCSSET)

0:2		Reserved	
3	HLDPLB	Hold PLB Transaction mode 0 1 PCI Express port completes any ongoing PLB transactions	Any new PLB transaction targeted to the PCI Express port is rearbitrated and any new inbound transaction is not processed, but is stored internally. The PLBIDL field (SDR0_PEx_RCSSTS[3]) indicates the completion of all ongoing PLB transactions.
4:6		Reserved	
7	RSTGU	GPL and UTL function Reset 0 1 Both the GPL and UTL functions are in the reset state	All the static configuration related to these functions (SDR0_PEx_UTLSETx) must retain stable after this field is set to 0. Any static configuration modification requires reset of the respective PCI Express port. A minimum of five PLB cycles must be preserved between the HLDPLB and RSTGU assertions. The RSTGU field must be kept set for at least four PLB cycles. Reset value = 1 (reset)
8:10		Reserved	
11	RDY	Ready 0 1 Indicates that the CFG function has completed its reset sequence and is ready to accept reads and writes	Inbound configuration requests addressed to an Endpoint will be replied with "Configuration Retry Status" (CRS) until this RDY field is set by the application.
12:14		Reserved	
15	RSTDL	Hot Reset (**Endpoint**) 0 1 After receiving the "Hot Reset" signal (TS1/TS2 ordered sets) requested by the remote Root Port, both the CFG and TL functions are in the reset state	**Endpoint:** This "Hot Reset" sequence is signalled to the Endpoint application by either polling the HRSTRQ field (PSDRn_RCSSTS[7] or by enabling the interrupt UIC2[2+n*8]. The PCLK250 clock must be switching for at least five full cycles before setting this field to 0. The "Hot Reset" causes the same functional reset as the AUX_RESET_ASYNC_BAR field (SDR0_PE0_PHY_CTL_RST[3]), with the exception of all sticky bits defined by the PCI Express specifications. **Root Port:** *Must never be set to 1.* Reset value = 1 (reset)
16:31		Reserved	

20.19.7 PCI Express n Reset, Clocking, and Shutdown Status (SDR0_PEn_RCSSTS)

Figure 20-37. PCI Express n Reset, Clocking and Shutdown Status Register (SDR0_PEn_RCSSTS)

0:2		Reserved	
3	PLBIDL	PLB Interface Idle 0 1 PCI Express port PLB interface is in the idle state	This means that any new PLB transaction targeted to the PCI Express port is rearbitrated and any new inbound transaction is no longer processed on the PLB interface, but is stored internally. It is recommended to enter a PCI Express Warm or Hot reset sequence *only* when the PCI Express PLB interface is in idle state.
4:6		Reserved	
7	HRSTRQ	Hot Reset (**Endpoint**) 0 When the "Hot Reset" signal is deasserted 1 Reports the receipt of the "Hot Reset" signal (TS1/TS2 ordered sets) requested by the remote Root Port	**Root Port:** Reserved
8:10		Reserved	
11	PGRST	PHYIF and DLP functions in Reset 0 Reports that the AUX_RESET_ASYNC_BAR field (SDR0_PE0_PHY_CTL_RST[3]) has been first set to 1 and then both the PHYIF and DLP functions have completed their initialization process 1 PHYIF and DLP functions of the PCI Express port are entering the reset state	To ensure proper resynchronization of the entire PCI Express port, the TL and CFG functions are automatically reset when this bit is asserted.
12:14		Reserved	
15	VC0ACT	VC0 Active 0 During inbound TLP reception from the TL function, the UTL function silently truncates and discards this TLP 1 Link is up and credits has been initialized	As long as this field is equal to 0, the UTL function silently discards all the outbound transactions issued by the system except local configuration cycles in the Root Port. It means the TL function requests initial credit update. After validating that all the receive buffers are empty and there are no completion-pending transmissions, the UTL function issues initial credit update to the TL function. After completing credits initialization, the TL function sets the VC0ACT field to 1 indicating that the TL and UTL function are ready to receive and transmit TLPs. If set to 0 during outbound TLP transmission, the UTL function silently discards the remainder of the TLP being transmitted and does not present any new TLPs for transmission until the VC0ACT field indication becomes high again.
16:22		Reserved	
23	BMEM	Bus Master Enable	This field is a copy of PECFGn_CMDSTATUS[2]). Because the latest write operation performed by the Root Port during the Endpoint PCI Express configuration is in the PECFGn_CMDSTATUS register, polling of the BMEN field by the Endpoint software allows the Endpoint to determine when the PCI Express Endpoint initialization is complete.
24:26		Reserved	

27	250OFF	PCI Express 250 MHz internal HSS Clock status 0 1 PCLK250 clock has been stopped because of the PHYIF function entering D2 power state	
28:31		Reserved	

20.19.8 PCI Express 0 Lane 0 BIST Register (SDR0_PE0_L0BIST)

Figure 20-38. PCI Express 0 Lane 0 BIST Register (SDR0_PE0_L0BIST)

0:6		Reserved	
7	BIST_Tx_RESET	Reset BIST pattern generator logic	
8:10		Reserved	
11	BIST_Rx_RESET	Reset BIST pattern checker logic	
12:14		Reserved	
15	BIST_BER_CLEAR	Clear BIST errors	
16:18		Reserved	
19	BIST_FORCE_ERR	Force BIST error in Tn data	
20:23	BIST_MODE	Select BIST mode / pattern 0000 Normal data 0001 K28.5 +/ (0011111010 1100000101 ...) 0010 D21.5 (1010101010 ….) 0011 K28.7 (001111100 ….) 0100 PCI Express compliance pattern (0011111010 1010101010 1100000101 0101010101...) 0101 D24.3 +/- (1100110011 0011001100 ...) 0110 All 0's 0111 All 1's 1000 PRBS7 (27-1) 1001 PRBS10 (210-1) 1010 PRBS15 (215-1) 1011 PRBS23 (223-1) 1100 PRBS31 (231-1)	
24:31		Reserved	

20.19.9 PCI Express 1 Lane n BIST Register (SDR0_PE1_LnBIST)

Figure 20-39. PCI Express 1 Lane n BIST Register (SDR0_PE1_LnBIST)

0:6		Reserved	
7	BIST_Tx_RESET	Reset BIST pattern generator logic	
8:10		Reserved	
11	BIST_Rx_RESET	Reset BIST pattern checker logic	
12:14		Reserved	
15	BIST_BER_CLEAR	Clear BIST errors	
16:18		Reserved	
19	BIST_FORCE_ERR	Force BIST error in Tn data	
20:23	BIST_MODE	Select BIST mode / pattern 0000 Normal data 0001 K28.5 +/ (0011111010 1100000101 ...) 0010 D21.5 (1010101010 ….) 0011 K28.7 (001111100 ….) 0100 PCI Express compliance pattern (0011111010 1010101010 1100000101 0101010101...) 0101 D24.3 +/- (1100110011 0011001100 ...) 0110 All 0's 0111 All 1's 1000 PRBS7 (27-1) 1001 PRBS10 (210-1) 1010 PRBS15 (215-1) 1011 PRBS23 (223-1) 1100 PRBS31 (231-1)	
24	XAUI_Ln_BYP_IN	Bypass Ln data into Tx	For SRIO only. Allows the digital block to input a serial signal (LX_SERDES_BYP_IN) directly to the driver.
25	XAUI_Ln_BYP_EN	Tx Bypass enable Ln	For SRIO only. The bypass mode (enabled by LX_SERDES_BYP_EN), allows the digital block to input a serial signal (LX_SERDES_BYP_IN) directly to the driver.
26:31		Reserved	

20.19.10 PCI Express x Lane n BIST Status Register (SDR0_PEx_LnBIST_STATUS)

Figure 20-40. PCI Express x Lane n BIST Status Register (SDR0_PEx_LnBIST_STATUS)

0:23		Reserved	
24	BIST_SYNC	BIST checker synchronized to incoming data	
25	BIST_TOG	Toggling state with each BIST error	
26	BIST_ERR	One or more BIST error in Rx data	
27	BIST_PATTERN_SYNC	First word in PRBS pattern	
28:31	BIST_ERR_CNT	Number of errors in Rx Data	

20.19.11 PCI Express 0 Lane 0 CDR Control Register (SDR0_PE0_L0CDRCTL)

Figure 20-41. PCI Express 0 Lane 0 CDR Control Register (SDR0_PE0_L0CDRCTL)

0:17		Reserved	
18:19	CDR_MODE	Select level of 1st & 2nd order control loop dampening 00 Reserved 01 Set for ppm offset > 3000 ppm 10 Reserved 11 Set for ppm offset < 1000 ppm	Reset value = 0b11.
20:21		Reserved	
22:23	CDR_THRESHOLD	Select threshold of clock recovery filter 00 High 01 Medium high (recomended setting for asynchronous SATA system) 10 Medium low (recomended setting for asynchronous PCI Express system) 11 Low	
24:25		Reserved	
26:27	CDR_FREQLOOP_GAIN	Set CDR 2nd order loop gain 00 High 01 Medium high (recomended setting for asynchronous SATA system) 10 Medium low 11 Low (recomended setting for asynchronous PCI Express system)	
28:30		Reserved	
31	CDR_FREQLOOP_EN	Enable CDR 2nd order loop 0 For synchronous system (Same reference clock for both ends of the link) 1 For asynchronous system (Different reference clocks on each end of the link)	Frequency Offset / CDR_FREQ LOOP_EN / CDR_FREQ LOOP_GAIN / CDR_THRE SHOLD 0 ppm (sync system) / 0 / xx / 11 < 600 ppm (PCI Express) / 1 / 11 / 10 < 1000 ppm (SATA) / 1 / 01 / 01

20.19.12 PCI Express 1 Lane n CDR Control Register (SDR0_PE1_LnCDRCTL)

Figure 20-42. PCI Express 1 Lane n CDR Control Register (SDR0_PE1_LnCDRCTL)

0:2		Reserved	
3	CDR_DN	Move the phase of the recovered clock down one step	For SRIO only. CDR_FREEZE field needs to be asserted (test purpose only).
4:6		Reserved	
7	CDR_UP	Move the phase of the recovered clock up one step	For SRIO only. CDR_FREEZE field needs to be asserted (test purpose only).
8:10		Reserved	
11	CDR_FREEZE	Freeze the phase of the clock recovery circuit	For SRIO only. Both CDR_DN & CDR_UP fields need to be deasserted to freeze the phase (test purpose only).
12:14		Reserved	
15	CDR_RESET	Reset the clock recovery circuit.	
16:17		Reserved	
18:19	CDR_MODE	Select level of 1st & 2nd order control loop dampening 00 Reserved 01 Set for ppm offset > 3000 ppm 10 Reserved 11 Set for ppm offset < 1000 ppm	Reset value = 0b11.
20:21		Reserved	
22:23	CDR_THRESHOLD	Select threshold of clock recovery filter 00 High 01 Medium high 10 Medium low (recomended setting for asynchronous PCI Express system) 11 Low (recomended setting for synchronous system or asynchronous SRIO system)	
24:25		Reserved	
26:27	CDR_FREQLOOP_GAIN	Set CDR 2nd order loop gain 00 High (recomended setting for synchronous system) 01 Medium high 10 Medium low 11 Low (recomended setting for asynchronous PCI Express and SRIO system)	
28:30		Reserved	
31	CDR_FREQLOOP_EN	Enable CDR 2nd order loop 0 For synchronous system (Same reference clock for both ends of the link) 1 For asynchronous system (Different reference clocks on each end of the link)	Frequency Offset / CDR_FREQ LOOP_EN / CDR_FREQ LOOP_GAIN / CDR_THRE SHOLD 0 ppm (sync system) / 0 / xx / 11 < 200 ppm (SRIO) / 1 / 11 / 11 < 600 ppm (PCI-Express) / 1 / 11 / 10

20.19.13 PCI Express 0 Lane 0 Drive Register (SDR0_PE0_L0DRV)

Figure 20-43. PCI Express 0 Lane 0 Drive Register (SDR0_PE0_L0DRV)

0:9		Reserved	
10:19	DRV_MODE	Driver output mode for debug purpose	For PCI Express only. Bits 10:18 Reserved for future enhancements Bit 19 Tx Bandwidth - Enable high bandwidth mode
20:22		Reserved	
23:27	DRV_LVL	Select the drive level on the output drive signal swing 00000 0 mV 00001 25 mV 00010 50 mV 00011 75 mV 00100 100 mV 00101 125 mV 00110 150 mV 00111 175 mV 01000 200 mV 01001 225 mV 01010 250 mV 01011 275 mV 01100 300 mV 01101 325 mV 01110 350 mV 01111 375 mV 10000 400 mV 10001 425 mV 10010 450 mV 10011 475 mV 10100 – 11111 Reserved	
28		Reserved	
29:31	EMP_POST	Set the de-emphasis level on the output drive signal swing 000 0.0 mV 001 12.5 mV 010 25.0 mV 011 37.5 mV 100 50.0 mV 101 62.5 mV 110 75.0 mV 111 87.5 mV	

20.19.14 PCI Express 1 Lane n Drive Register (SDR0_PE1_LnDRV)

Figure 20-44. PCI Express 1 Lane n Drive Register (SDR0_PE1_LnDRV)

0:2		Reserved	
3	DRV_EN	Drive enable 0 Disable 1 Enable	For SRIO only. Reset value = 1.
4:6		Reserved	
7	TX_HIGHZ	Select transmitter high impedance mode 0 Normal mode 1 Transmitter in high impedance	For SRIO only.
8:9		Reserved	
10:19	DRV_MODE	Driver output mode for debug purpose	Bits 10:18 Reserved for future enhancements Bit 19 Tx Bandwidth - Enable high bandwidth mode
20:22		Reserved	
23:27	DRV_LVL	Select the drive level on the output drive signal swing 00000 0 mV 00001 25 mV 00010 50 mV 00011 75 mV 00100 100 mV 00101 125 mV 00110 150 mV 00111 175 mV 01000 200 mV 01001 225 mV 01010 250 mV 01011 275 mV 01100 300 mV 01101 325 mV 01110 350 mV 01111 375 mV 10000 400 mV 10001 425 mV 10010 450 mV 10011 475 mV 10100 – 11111 Reserved	
28		Reserved	
29:31	EMP_POST	Set the de-emphasis level on the output drive signal swing 000 0.0 mV 001 12.5 mV 010 25.0 mV 011 37.5 mV 100 50.0 mV 101 62.5 mV 110 75.0 mV 111 87.5 mV	

20.19.15 PCI Express 0 Lane 0 Receiver Register (SDR0_PE0_L0REC)

Figure 20-45. PCI Express 0 Lane 0 Receiver Register (SDR0_PE0_L0REC)

0:2		Reserved	
3	RXPOLARITY	Receive data polarity 0 No polarity inversion of the received data 1 Polarity inversion of the received data	For SATA only.
4:5		Reserved	
6:7	RX_SIGDET_LVL	Select receiver signal detect level 00 Min - 50 mV ppd Max - 150 mV ppd (for SATA) 01 Min - 60 mV ppd Max - 175 mV ppd (for SATA) 10 Min - 75 mV ppd Max - 200 mV ppd (SATA I/II i, m) 11 Min - 120 mV ppd Max - 240 mV ppd (SATA I/II x)	Reset value = 0b10.
8:11		Reserved	
12:15	RX_EQ	Receiver equalization level 0000 Off 0010 Low 0110 Medium Low 0111 Medium 1111 High	For SATA only.
16:31		Reserved	

20.19.16 PCI Express 1 Lane n Receiver Register (SDR0_PE1_LnREC)

Figure 20-46. PCI Express 1 Lane n Receiver Register (SDR0_PE1_LnREC)

0:5		Reserved	
6:7	RX_SIGDET_LVL	Select receiver signal detect level 00 Min – 50 mV ppd Max – 150 mV ppd 01 Min – 60 mV ppd Max – 175 mV ppd 10 Min – 75 mV ppd Max – 200 mV ppd 11 Min – 120 mV ppd Max – 240 mV ppd	Reset value = 0b10.
8:10		Reserved	
11	RX_SIGDET_EN	Enable receiver detect circuit 0 Receiver signal detect circuit disabled 1 Normal operation – BIAS_EN should be also asserted to provide proper bias	For SRIO only. Reset value = 1.
12:15	RX_EQ	Receiver equalization level 0000 Off 0010 Low 0110 Medium Low 0111 Medium 1111 High	For SRIO only.
16:18		Reserved	
19	RX_HIGHZ	Select receiver high impedance mode 0 Normal mode 1 Receiver in high impedance	For SRIO only.
20:22		Reserved	
23	RX_CM	Common mode voltage 0 Common mode voltage set externally 1 Common mode voltage set internally - Must be set when using either internal or external AC coupling capacitor	For SRIO only.
24:26		Reserved	
27	RX_AC_COUPLE	On-chip coupling mode 0 DC coupling 1 AC coupling (RX_CM must be asserted Only enabled for > 2.5 Gbps SRIO)	For SRIO only.
28:30		Reserved	
31	RX_TERM	Termination mode 0 Common mode set to ground 1 Common mode set to VDD/2	For SRIO only.

Table 20-14. Rx Termination Modes

Rx Termination Mode	RX_TERM	RX_AC_COUPLE	RX_CM	RX_HIGHZ
PCI Express termination- center tap to GND	0	1	1	0
DC coupled	1	0	0	0
AC coupled - Off chip	1	0	1	0
AC coupled - On chip	1	1	1	0
HZ Tx requires Lx_DRV_EN=0	0	0	0	1
All others are illegal				

20.19.17 PCI Express 0 Lane 0 Loopback Register (SDR0_PE0_L0LPBK)

For Lane 0 only.

Figure 20-47. PCI Express 0 Lane 0 Loopback Register (SDR0_PE0_L0LPBK)

0:8		Reserved	
9:11	REC_DETECT_USEC	Indicate the appropriate period of time to delay before sampling the receiver detection circuitry in µs. 000 1 001 2 010 4 011 5 100 10 101 20 110 40 111 50	Reset value = 0b010.
12:15	PHY_LATENCY	Define the number of PClk cycles the electrical idle signal should be delayed to match PHY architecture	For PCI Express only. All other values are reserved. Reset value = 0b0111.
16:26		Reserved	
27	LINE_LPB_EN ANALOGB (SATA)	Enable line loop back	Internal Rx to Tx (non retimed).
28:30		Reserved	
31	SER_LPB_EN	Enable serial loop back - Internal Rx to Tx (non retimed)	For PCI Express only. Internal Tx to Rx loopback.

20.19.18 PCI Express 1 Lane 0 Loopback Register (SDR0_PE1_L0LPBK)

For Lane 0 only.

Figure 20-48. PCI Express 1 Lane 0 Loopback Register (SDR0_PE1_L0LPBK)			
0:8		Reserved	
9:11	REC_DETECT_USEC	Indicate the appropriate period of time to delay before sampling the receiver detection circuitry µs 000 1 001 2 010 4 011 5 100 10 101 20 110 40 111 50	Reset value = 0b010.
12:15	PHY_LATENCY	Define the number of PClk cycles the electrical idle signal should be delayed to match PHY architecture	For PCI Express only. All other values are reserved. Reset value = 0b0111.
16:18		Reserved	
19	RCLK_LPB_EN	Enable recovered clock loopback to driver	For SRIO only.
20:22		Reserved	
23	PAR_LPB_EN	Enable parallel loop back	For SRIO only.
24:26		Reserved	
27	LINE_LPB_EN ANALOGB (SATA)	Enable line loop back	Internal Rx to Tx (non retimed).
28:30		Reserved	
31	SER_LPB_EN	Enable serial loop back - Internal Rx to Tx (non retimed)	Internal Tx to Rx loopback.

20.19.19 PCI Express 1 Lane n Loopback Register (SDR0_PE1_LnLPBK)

For Lanes 1–3.

Figure 20-49. PCI Express 1 Lane n Loopback Register (SDR0_PE1_LnLPBK)			
0:18		Reserved	
19	RCLK_LPB_EN	Enable recovered clock loopback to driver	For SRIO only.
20:22		Reserved	
23	PAR_LPB_EN	Enable parallel loop back	For SRIO only.
24:26		Reserved	
27	LINE_LPB_EN ANALOGB (SATA)	Enable line loop back	Internal Rx to Tx (non retimed).
28:30		Reserved	
31	SER_LPB_EN	Enable serial loop back - Internal Rx to Tx (non retimed)	Internal Tx to Rx loopback.

20.19.20 PCI Express 0 Lane 0 Clocking Register (SDR0_PE0_L0CLK)

Figure 20-50. PCI Express 0 Lane 0 Clocking Register (SDR0_PE0_L0CLK)

0:6		Reserved	
7	SYSCLK_DIV2_SEL	Reference clock at half the required rate	For PCI Express only. Should not be used in operational mode (only for test purpose)
8:28		Reserved	
29:31	SYSCLK_RATE	Select PLL multiplier ratio PCI Express Gen 1: 000 Multiply by 8.0 - Ref. clock = 312.50 MHz. 001 Multiply by 10.0 - Ref. clock = 250.00 MHz. 010 Multiply by 12.5 - Ref. clock = 200.00 MHz. 011 Multiply by 15.0 - Ref. clock = 166.67 MHz. 100 Multiply by 16.0 - Ref. clock = 156.25 MHz. 101 Multiply by 20.0 - Ref. clock = 125.00 MHz. 110 Multiply by 25.0 - Ref. clock = 100.00 MHz. 111 Multiply by 30.0 - Ref. clock = 83.33 MHz. SATA: 000 Multiply by 8.0 - Ref. clock = 375.0 MHz. 001 Multiply by 10.0 - Ref. clock = 300.0 MHz. 010 Multiply by 12.5 - Ref. clock = 240.0 MHz. 011 Multiply by 15.0 - Ref. clock = 200.0 MHz. 100 Multiply by 16.0 - Ref. clock = 187.5 MHz. 101 Multiply by 20.0 - Ref. clock = 150.0 MHz. 110 Multiply by 25.0 - Ref. clock = 120.0 MHz. 111 Multiply by 30.0 - Ref. clock = 100.0 MHz.	Reset value = 0b101.

20.19.21 PCI Express 1 Lane 0 Clocking Register (SDR0_PE1_L0CLK)

For Lane 0 only.

Figure 20-51. PCI Express 1 Lane 0 Clocking Register (SDR0_PE1_L0CLK)

0:6		Reserved	
7	SYSCLK_DIV2_SEL	Reference clock at half the required rate	Should not be used in operational mode (only for test purpose)
8:10		Reserved	
11	CLKBUF_EN	Enable clock buffers 0 Clock buffer powered down 1 Normal operation	For SRIO only. Reset value = 1.
12:14		Reserved	
15	RX_BAND	Selection of the receive lane rate 0 Full rate (2.12–3.2 Gbps) 1 Half rate (1.06–1.6 Gbps)	For SRIO only.
16:18		Reserved	
19	TX_BAND	Selection of the transmit lane rate 0 Full rate (2.12–3.2 Gbps) 1 Half rate (1.06–1.6 Gbps)	For SRIO only.
20:22		Reserved	

23	SYSCLK_EN	Reference clock receiver mode 0 Reference Clock receiver inactive - All clock buffers are powered down 1 Reference Clock receiver active and ready for use with external clock	For SRIO only. Reset value = 1.
24:26		Reserved	
27	SYSCLK_TERM_SEL	On Die 100 ohms on die termination for the Reference clock 0 On die termination disabled 1 On die termination enabled	For SRIO only.
28		Reserved	
29:31	SYSCLK_RATE	Select PLL multiplier ratio PCI Express Gen 1: 000 Multiply by 8.0 - Ref. clock = 312.50 MHz. 001 Multiply by 10.0 - Ref. clock = 250.00 MHz. 010 Multiply by 12.5 - Ref. clock = 200.00 MHz. 011 Multiply by 15.0 - Ref. clock = 166.67 MHz. 100 Multiply by 16.0 - Ref. clock = 156.25 MHz. 101 Multiply by 20.0 - Ref. clock = 125.00 MHz. 110 Multiply by 25.0 - Ref. clock = 100.00 MHz. 111 Multiply by 30.0 - Ref. clock = 83.33 MHz. SRIO (3.125/2.5 Gbps): 000 Multiply by 8.0 - Ref. clock = 390.6250/312.50 MHz. 001 Multiply by 10.0 - Ref. clock = 312.5000/250.00 MHz. 010 Multiply by 12.5 - Ref. clock = 250.0000/200.00 MHz. 011 Multiply by 15.0 - Ref. clock = 208.3333/166.67 MHz. 100 Multiply by 16.0 - Ref. clock = 195.3125/156.25 MHz. 101 Multiply by 20.0 - Ref. clock = 156.1250/125.00 MHz. 110 Multiply by 25.0 - Ref. clock = 125.0000/100.00 MHz. 111 Multiply by 30.0 - Ref. clock = 104.1667/83.33 MHz.	Reset value = 0b101.

20.19.22 PCI Express 1 Lane n Clocking Register (SDR0_PE1_LnCLK)

For Lanes 1–3.

Figure 20-52. PCI Express 1 Lane n Clocking Register (SDR0_PE1_LnCLK)

0:10		Reserved	
11	CLKBUF_EN	Enable clock buffers 0 Clock buffer powered down 1 Normal operation	For SRIO only. Reset value = 1.
12:14		Reserved	
15	RX_BAND	Selection of the receive lane rate 0 Full rate (2.12 - 3.2 Gbps) 1 Half rate (1.06 - 1.6 Gbps)	For SRIO only.
16:18		Reserved	

19	TX_BAND	Selection of the transmit lane rate 0 Full rate (2.12 - 3.2 Gbps) 1 Half rate (1.06 - 1.6 Gbps)	For SRIO only.
20:31		Reserved	

20.19.23 PCI Express 0 PHY Control Reset Register (SDR0_PE0_PHY_CTL_RST)

Figure 20-53. PCI Express 0 PHY Control Reset Register (SDR0_PE0_PHY_CTL_RST)

0:2		Reserved	
3	AUX_RESET_ASYNC_BAR	PCI Express PHY Reset 0 PCI Express port up to the TL function is reset 1 PCI Express stack reset exit sequence occurs	For PCI Express only. Active low signal - Required to be active while another protocol than PCI Express is applied by the PHY. To ensure proper resynchronization of the entire PCI Express port, both TL and CFG as well as the HSS functions are automatically reset if this bit is asserted. The PCI Express PLL LC tank must be running at its required frequency before setting the AUX_RESET_ASYNC_BAR field to 1.
4:27		Reserved	
28	PCLK_REQ	Restart PLL and PCLK	PCLK and PLL restart (PCI Express) Supporting the Wireless Form Factor extension to PCI Express, this will re-enable the PLL and restart PCLK in the P1 state when high in the P1 power state. When low, this signal is ignored.
29	PCLK_OFF	Disable PCLK	In support for the Wireless Form Factor extension to PCI Express, this allows the user to disable the PCLK in the P1 power down state. The PLL is disabled to further reduce power for WFF PCI Express applications.
30:31	PROTOCOL	Select PHY protocol mode 00 PCI Express 1.1 compliant mode 01 SATA I/II compliant mode 10 Reserved 11 Reserved	Reset value = 0b0, SDR0_SDSTP1[SATA]

20.19.24 PCI Express 1 PHY Control Reset Register (SDR0_PE1_PHY_CTL_RST)

Figure 20-54. PCI Express 1 PHY Control Reset Register (SDR0_PE1_PHY_CTL_RST)

0	RESET_TSYNC	Reset transmitter synchronization logic	For SRIO only. This active high signal must be toggled (0-1-0) any time the lane's transmit data rate is changed.
1:2		Reserved	

3	AUX_RESET_ASYNC_BAR	PCI Express PHY Reset 0 PCI Express port up to the TL function is reset 1 PCI Express stack reset exit sequence occurs	For PCI Express only. Active low signal - Required to be active while another protocol than PCI Express is applied by the PHY. The PCI Express port up to the TL function is reset. To ensure proper resynchronization of the entire PCI Express port, both TL and CFG as well as the HSS functions are automatically reset if this bit is asserted. The PCI Express PLL LC tank must be running at its required frequency before setting the AUX_RESET_ASYNC_BAR field to 1.
4	RX_EN[3]	Enable receiver path Lane 3	For SRIO only. Reset value = 1 for each bit.
5	RX_EN[2]	Enable receiver path Lane 2	
6	RX_EN[1]	Enable receiver path Lane 1	
7	RX_EN[0]	Enable receiver path Lane 0	
8	TX_EN[3]	Enable transmitter path Lane 3	
9	TX_EN[2]	Enable transmitter path Lane 2	
10	TX_EN[1]	Enable transmitter path Lane 1	
11	TX_EN[0]	Enable transmitter path Lane 0	
12:13		Reserved	
14	RES_CAL	Initialize resistor calibration sequence	For SRIO only.
15	VCO_CAL	Initialize VCO calibration sequence	
16	RESET_TSYNC_EN[3]	Enable reset transmitter sync when change rate L3	For SRIO only. When enabled (high), RESET_TSYNC is applied to lane to resynchronize transmitter serialization pathway. When disabled (low), RESET_TSYNC is ignored.
17	RESET_TSYNC_EN[2]	Enable reset transmitter sync when change rate L2	
18	RESET_TSYNC_EN[1]	Enable reset transmitter sync when change rate L1	
19	RESET_TSYNC_EN[0]	Enable reset transmitter sync when change rate L0	
20	RESET_RSYNC[3]	Reset receiver sync L3	For SRIO only. Reset receiver synchronization logic. This active high signal must be toggled (0-1-0) any time the lane's receive data rate is changed.
21	RESET_RSYNC[2]	Reset receiver sync L2	
22	RESET_RSYNC[1]	Reset receiver sync L1	
23	RESET_RSYNC[0]	Reset receiver sync L0	
24	BIAS_EN	Enable bias generator 0 Bias generator disabled and powered down - All side clocks except C_SYSCLK_OUT are disabled 1 Normal operation	For SRIO only. Reset value = 1.
25	CLKBUF_R_EN	Enable clock distribution on Lane 2-3	For SRIO only. Reset value = 1.
26	CLKBUF_L_EN	Enable clock distribution on Lane 0-1	For SRIO only. Reset value = 1.
27	CMU_EN	Enable PLL 0 PLL disabled and powered down - All side clocks except C_SYSCLK_OUT are disabled 1 Normal operation	For SRIO only. Reset value = 1.

28	PCLK_REQ	Restart PLL and PCLK	PCLK and PLL restart (PCI Express) Supporting the Wireless Form Factor extension to PCI Express, this will re-enable the PLL and restart PCLK in the P1 state when HIGH in the P1 power state. When LOW, this signal is ignored.
29	PCLK_OFF	Disable PCLK	In support for the Wireless Form Factor extension to PCI Express, this allows the user to disable the PCLK in the P1 power down state. The PLL is disabled to further reduce power for WFF PCI Express applications.
30:31	PROTOCOL	Select PHY protocol mode 00 PCI Express 1.1 compliant mode 01 Reserved 10 SRIO compliant mode 11 Reserved	Reset value = SDR0_SDSTP1[SRIO], 1b0

20.19.25 PCI Express 0 Reset Status Register (SDR0_PE0_RSTSTA)

Figure 20-55. PCI Express 0 Reset Status Register (SDR0_PE0_RSTSTA)

0:16		Reserved	
17	C_RESET_SATA	PHY reset status	For SATA only.
18	POR_RESET_BAR	SATA PHY reset	For SATA only. Active low signal - Require to be active while another protocol than SATA is applied to the PHY.
19:25		Reserved	
26	CMULOCK	Indication that the analog PHY PLL is locked to the reference clock	For PCI Express, this information is also provided as PLLLK field = SDR0_PEx_RCSSTS[19].
27		Reserved	
28	C_READY_PCI	Indication that the analog PHY block is ready for operation (PMA layer initialized)	For PCI Express only.
29	C_READY_SATA	SATA ready for set up sequence	For SATA only.
30		Reserved	
31	C_PHYRESET_BAR	PHY reset status	For PCI Express only. Active low status driven during power-on reset and before Px_PxClk is turned off.

20.19.26 PCI Express 1 Reset Status Register (SDR0_PE1_RSTSTA)

Figure 20-56. PCI Express 1 Reset Status Register (SDR0_PE1_RSTSTA)

0:18		Reserved	
19	C_RESET_XAUI	SRIO PHY reset 0 Normal operation for SRIO 1 Reset	For SRIO only. Set to 1 when PCIe port 1 is used. Reset value = 1.
20:23		Reserved	

24	RES_CAL_DONE	Indication that the resistor calibration process is completed	For SRIO only.
25	VCO_CAL_DONE	Indication that the VCO calibration process is completed	
26	CMULOCK	Indication that the analog PHY PLL is locked to the reference clock	For PCI Express, this information is also provided as PLLLK field = SDR0_PEx_RCSSTS[19].
27		Reserved	
28	C_READY_PCI	Indication that the analog PHY block is ready for operation (PMA layer initialized)	For PCI Express only.
29		Reserved	
30	C_READY_XAUI	SRIO ready for set up sequence	For SRIO only.
31	C_PHYRESET_BAR	PHY reset status	For PCI Express only. Active low status driven during power-on reset and before Px_PxClk is turned off.

20.19.27 PCI Express 0 Observation Register (SDR0_PE0_OBS)

Figure 20-57. PCI Express 0 Observation Register (SDR0_PE0_OBS)

0	PHYSTATUS	PCI Express PHY Status	For PCI Express only.
1:15		Reserved	
16:19	RxSTATE	SATA Rx State	For SATA only.
20:23	TxSTATE	SATA Tx State	
24:27	PHYSTATE	SATA PHY State	
28:29	WSTATE	PCI Express WakeFSM State Machine	For PCI Express only.
30:31	PSTATE	PCI Express MainFSM State Machine	

20.19.28 PCI Express 1 Observation Register (SDR0_PE1_OBS)

Figure 20-58. PCI Express 1 Observation Register (SDR0_PE1_OBS)

0	PHYSTATUS	PCI Express PHY Status	For PCI Express only.
1:11		Reserved	
12:15	TX_HIGHZ	Select transmitter high impedance mode Normal mode Transmitter in high impedance	For SRIO only.
16:27		Reserved	
28:29	WSTATE	PCI Express WakeFSM State Machine	For PCI Express only.
30:31	PSTATE	PCI Express MainFSM State Machine	

20.19.29 PCI Express 0 PHY Test Register (SDR0_PE0_PHY_TEST)

Figure 20-59. PCI Express 0 PHY Test Register (SDR0_PE0_PHY_TEST)

0	TEST_EN	Enable Test Controls	For PCI Express only.
1	PERST_BAR	PERST_BAR Test Input	For PCI Express only. Enabled when TEST_EN = 1. Reset value = 1.
2:3	POWERDOWN	POWERDOWN Test Input	For PCI Express only. Enabled when TEST_EN = 1.
4:18		Reserved	
19	TXDETRXLOOP	TXDETRXLOOP Test Input	For PCI Express only. Enabled when TEST_EN = 1.
20:22		Reserved	
23	TXELECIDLE	TXELECIDLE Test Input	For PCI Express only. Enabled when TEST_EN = 1.
24:26		Reserved	
27	TXCOMPLIANCE	TXCOMPLIANCE Test Input	For PCI Express only. Enabled when TEST_EN = 1.
28:30		Reserved	
31	RXPOLARITY	RXPOLARITY Test Input	For PCI Express only. Enabled when TEST_EN = 1.

20.19.30 PCI Express 1 PHY Test Register (SDR0_PE1_PHY_TEST)

Figure 20-60. PCI Express 1 PHY Test Register (SDR0_PE1_PHY_TEST)

0	TEST_EN	Enable Test Controls	For PCI Express only.
1	PERST_BAR	PERST_BAR Test Input	For PCI Express only. Enabled when TEST_EN = 1. Reset value = 1.
2:3	POWERDOWN	POWERDOWN Test Input	For PCI Express only. Enabled when TEST_EN = 1.
4:18		Reserved	
19	TXDETRXLOOP	TXDETRXLOOP Test Input	For PCI Express only. Enabled when TEST_EN = 1.
20	TXELECIDLE[3]	TXELECIDLE L3 Test Input	For PCI Express only. Enabled when TEST_EN = 1.
21	TXELECIDLE[2]	TXELECIDLE L2 Test Input	
22	TXELECIDLE[1]	TXELECIDLE L1 Test Input	
23	TXELECIDLE[0]	TXELECIDLE L0 Test Input	
24	TXCOMPLIANCE[3]	TXCOMPLIANCE L3 Test Input	For PCI Express only. Enabled when TEST_EN = 1.
25	TXCOMPLIANCE[2]	TXCOMPLIANCE L2 Test Input	
26	TXCOMPLIANCE[1]	TXCOMPLIANCE L1 Test Input	
27	TXCOMPLIANCE[0]	TXCOMPLIANCE L0 Test Input	

28	RXPOLARITY[3]	RXPOLARITY L3 Test Input	For PCI Express only. Enabled when TEST_EN = 1.
29	RXPOLARITY[2]	RXPOLARITY L2 Test Input	
30	RXPOLARITY[1]	RXPOLARITY L1 Test Input	
31	RXPOLARITY[0]	RXPOLARITY L0 Test Input	

20.19.31 PCI Express x Lane n Error Counter (SDR0_PEx_LnERRC)

Figure 20-61. PCI Express x Lane n Clocking Register (SDR0_PEx_LnERRC)

0:15		Reserved	
16:31	ERR_CNT	Error Counter	

20.19.32 PCI Express Interrupt Handler Interface Setting 1 Register (SDR0_PEIHS1)

Figure 20-62. PCI Express Interrupt Handler Interface Setting 1 Register (SDR0_PEIHS1)

0:31	BASEH	PCI Express Interrupt Handler Base Address High	Reset value = 0x0000_000C.

20.19.33 PCI Express Interrupt Handler Interface Setting 2 Register (SDR0_PEIHS2)

Figure 20-63. PCI Express Interrupt Handler Interface Setting 2 Register (SDR0_PEIHS2)

0:23	BASEL	PCI Express Interrupt Handler Base Address Low	Reset value = 0x1000_00.
24:28		Reserved	
29	OMBR	Outbound Message Byte Reverse 0 No byte reversal 1 Reverse bytes	Outbound message data from PIH to PCI Express. Message data is normally byte reversed for endianess conversion (little-endian to big-endian). Setting OMBR would put the bytes back in order (little-endian).
30	IMBR	Inbound Message Byte Reverse 0 No byte reversal 1 Reverse bytes	Inbound message data snooped during PLB Bus write. Message data is normally byte reversed by endianess conversion (PCI little-endian to PLB big-endian). Setting IMBR would do the compare with little-endian data.
31	RESET	PCI Express Interrupt Handler Reset	Reset value = 1.

20.20 PCI Express DCR Registers (PEGPLn_xxxx)

The DCR Register Block is mapped into the DCR bus address space. Each PCI Express port consists of 32 DCR addresses. The DCR base address for each PCI Express port is defined as follows:

- PCI Express port PE0: DCR base address is 0x100
- PCI Express port PE1: DCR base address is 0x120

These registers enable the controlling and monitoring of the PCI Express PLB Slave interface, with the exception of the GPL Configuration Register (PEGPLn_CFG - page 564), which contains some bits that are related to the PCI Express PLB Master interface.

Table 20-15 shows the DCR address offset mapping for each PCI Express port. The letter n in the PEGPLn prefix of each register name represents either 0 or 1, depending on the PCI Express port to be addressed.

All of the registers listed in the following table have a reset value of 0.

Table 20-15. PCI Express DCRs

Mnemonic	Register	DCR Offset	Access	Page
PEGPLn_CFGBAH	Configuration Region Base Address High	0x00	R/W	556
PEGPLn_CFGBAL	Configuration Region Base Address Low	0x01	R/W	557
PEGPLn_CFGMSK	Configuration Region Mask and Validity	0x02	R/W	557
PEGPLn_MSGBAH	Message Region Base Address High	0x03	R/W	557
PEGPLn_MSGBAL	Message Region Base Address Low	0x04	R/W	558
PEGPLn_MSGMSK	Message Region Mask and Validity	0x05	R/W	558
PEGPLn_OMR1BAH	Outbound Memory Region 1 Base Address High	0x06	R/W	558
PEGPLn_OMR1BAL	Outbound Memory Region 1 Base Address Low	0x07	R/W	558
PEGPLn_OMR1MSKH	Outbound Memory Region 1 Mask High	0x08	R/W	559
PEGPLn_OMR1MSKL	Outbound Memory Region 1 Mask and Validity Low	0x09	R/W	559
PEGPLn_OMR2BAH	Outbound Memory Region 2 Base Address High	0x0A	R/W	559
PEGPLn_OMR2BAL	Outbound Memory Region 2 Base Address Low	0x0B	R/W	560
PEGPLn_OMR2MSKH	Outbound Memory Region 2 Mask High	0x0C	R/W	560
PEGPLn_OMR2MSKL	Outbound Memory Region 2 Mask and Validity Low	0x0D	R/W	560
PEGPLn_OMR3BAH	Outbound Memory Region 3 Base Address High (BAR2H)	0x0E	R/W	560
PEGPLn_OMR3BAL	Outbound Memory Region 3 Base Address Low (BAR2L)	0x0F	R/W	561
PEGPLn_OMR3MSKH	Outbound Memory Region 3 Mask High	0x10	R/W	561
PEGPLn_OMR3MSKL	Outbound Memory Region 3 Mask and Validity Low	0x11	R/W	561
PEGPLn_REGBAH	Local Register Region Base Address High (BAR5H)	0x12	R/W	562
PEGPLn_REGBAL	Local Register Region Base Address Low (BAR5L)	0x13	R/W	562
PEGPLn_REGMSK	Local Register Region Mask and Validity	0x14	R/W	562
PEGPLn_SPECIAL	Special Purpose Prefetch Buffer Handler	0x15	R/W	563
PEGPLn_CFG	GPL Configuration Register	0x16	R/W	564
PEGPLn_ESR	Error Status Register	0x17	R/W	564
PEGPLn_EARH	Slave Error Address High Register	0x18	R	565
PEGPLn_EARL	Slave Error Address Low Register	0x19	R	565
PEGPLn_EATR	Slave Error Attribute Register	0x1A	R	565

20.20.1 Configuration Region Base Address High Register (PEGPLn_CFGBAH)

Functionally, the Configuration Region Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the Configuration Region (BAR4 for outbound non posted transactions).

Figure 20-64. Configuration Region Base Address High Register (PEGPLn_CFGBAH)

0:31	BAH	Configuration Region Base Address Bits (0:31)	

20.20.2 Configuration Region Base Address Low Register (PEGPLn_CFGBAL)

Figure 20-65. Configuration Region Base Address Low Register (PEGPLn_CFGBAL)			
0:24	BAL	Configuration Region Base Address Bits (32:56): These bits can be masked to define an address space from 128B to 4GB in size.	
25:31		Reserved	

20.20.3 Configuration Region Mask and Validity Register (PEGPLn_CFGMSK)

The size of the Configuration Region is programmable from 128 bytes to 4 GB by means of a mask vector. This vector is used to mask out the unnecessary address bits in the address decoder. Bits 0:31 of the Configuration Region Base Address Register (PEGPLn_CFGBAH - page 556) are always considered to be set, meaning not masked. Bits 32:56 can be masked. Bits 56:64 are always considered to be "do not care" in the address decoding process, meaning masked. Setting a bit in the mask vector to 1 allows it to be taken into consideration in the address decoding process, meaning not masked. The mask vector must be programmed as a *leading ones* vector, indicating that if one of the bits is set, all the previous MSBs must also be set.

Figure 20-66. Configuration Region Mask and Validity Register (PEGPLn_CFGMSK)			
0:24	BAM	Configuration Region Base Address Mask: Bits 32:56 of the Configuration Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 128B (2^7) to 4GB (2^{32}): 00000000000000000000000000 - 2^{32}B 10000000000000000000000000 - 2^{31}B 11000000000000000000000000 - 2^{30}B --- 11111111111111111111111100 - 2^9B 11111111111111111111111110 - 2^8B 11111111111111111111111111 - 2^7B	
25:30		Reserved	
31	VAL	Specifies whether the Configuration Region is valid: 0 The PCI Express PLB Slave interface does not respond to transactions that fall into this region.	

20.20.4 Message Region Base Address High Register (PEGPLn_MSGBAH)

Functionally, the Message Region Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the Message Region (BAR3 for outbound transactions).

Figure 20-67. Message Region Base Address High Register (PEGPLn_MSGBAH)			
0:31	BAH	Message Region Base Address Bits (0:31)	

20.20.5 Message Region Base Address Low Register (PEGPLn_MSGBAL)

Figure 20-68. Message Region Base Address Low Register (PEGPLn_MSGBAL)			
0:24	BAL	Message Region Base Address Bits 32:56: Bits 17:24 can be masked to define an address space from 128B to 32KB in size.	
25:31		Reserved	

20.20.6 Message Region Mask and Validity Register (PEGPLn_MSGMSK)

The size of the Message Region is programmable from 128 bytes to 32 KB by means of a mask vector. For details about the mask vector, see *Configuration Region Mask and Validity Register (PEGPLn_CFGMSK)* on page 557.

Figure 20-69. Message Region Mask and Validity Register (PEGPLn_MSGMSK)			
0:16		Reserved	
17:24	BAM	Message Region Base Address Mask: Bits 49:56 of the Message Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 128B (2^7) to 32KB (2^{15}).	
25:30		Reserved	
31	VAL	Specifies whether the Message Region is valid: 0 The PCI Express PLB Slave interface does not respond to transactions that fall into this region.	

20.20.7 Outbound Memory Region 1 Base Address High Register (PEGPLn_OMR1BAH)

Functionally, the Outbound Memory Region 1 Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the first Outbound Memory Region (BAR0 for outbound transactions).

Figure 20-70. Outbound Memory Region 1 Base Address High Register (PEGPLn_OMR1BAH)			
0:31	BAH	Outbound Memory Region 1 Base Address Bits 0:31	

20.20.8 Outbound Memory Region 1 Base Address Low Register (PEGPLn_OMR1BAL)

Figure 20-71. Outbound Memory Region 1 Base Address Low Register (PEGPLn_OMR1BAL)			
0:4	BAL	Outbound Memory Region 1 Base Address Bits 32:36: BIts 1:36 can be masked to define an address space from 2^{27} (128MB) to 2^{63} in size.	
5:31		Reserved	

20.20.9 Outbound Memory Region 1 Mask High Register (PEGPLn_OMR1MSKH)

Functionally, the Outbound Memory Region 1 Mask High and Low Registers can be viewed as a single 64-bit register that defines the size of this address region. These registers must be programmed such that the concatenated mask is in the form of *leading ones*, meaning that if one of the bits is set, all the previous MSBs must also be set. The size of the Outbound Memory Region 1 is programmable from 2^{27}(128 MB) to 2^{63} bytes by means of a mask vector.

Figure 20-72. Outbound Memory Region 1 Mask High Register (PEGPLn_OMR1MSKH)

0		Reserved	
1:31	BAMH	Outbound Memory Region 1 Mask Bits (1:31): Bits 1:36 of the Base Address can be masked. This region can range in size from 2^{27} (128MB) to 2^{63}. The mask defines the bits that are considered by the address decoder, thus determining the size of the region: 00000000000000000000000000000000000 - 2^{63}B 10000000000000000000000000000000000 - 2^{62}B 11000000000000000000000000000000000 - 2^{61}B --- 11111111111111111111111111111111100 - 2^{29}B 11111111111111111111111111111111110 - 2^{28}B 11111111111111111111111111111111111 - 2^{27}B	

20.20.10 Outbound Memory Region 1 Mask and Validity Low Register (PEGPLn_OMR1MSKL)

Figure 20-73. Outbound Memory Region 1 Mask and Validity Low Register (PEGPLn_OMR1MSKL

0:4	BAML	Outbound Memory Region 1 Mask Bits (32:36). See explanation for PEGPLn_OMR1MSKH[BAMH].	
5:21		Reserved	
22:29	SRV	Sub-Region Valid: The Outbound Memory Region 1 is divided into eight identical sub-regions. This signal defines whether the Nth sub-region is valid (Bit 22 enables sub-region 0, bit 23 enables sub-region 1, and so on). If the sub-region enable bit is set to 0, the PCI Express PLB Slave interface does not respond to transactions that fall into this memory sub-region. If the UOT bit in this register is set to 1, this field is ignored.	
30	UOT	Use One TC: 1 Outbound Memory Region 1 is not divided into eight sub-regions, but is used as a single wide memory area.	
31	VAL	Specifies whether Outbound Memory Region 1 is valid: 0 The PCI Express PLB Slave interface does not respond to transactions that fall into this region.	

20.20.11 Outbound Memory Region 2 Base Address High Register (PEGPLn_OMR2BAH)

Functionally, the Outbound Memory Region 2 Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the second Outbound Memory Region (BAR1 for outbound transactions).

Figure 20-74. Outbound Memory Region 2 Base Address High Register (PEGPLn_OMR2BAH)			
0:31	BAH	Outbound Memory Region 2 Base Address Bits (0:31)	

20.20.12 Outbound Memory Region 2 Base Address Low Register (PEGPLn_OMR2BAL)

Figure 20-75. Outbound Memory Region 2 Base Address Low Register (PEGPLn_OMR2BAL)			
0:11	BAL	Outbound Memory Region 2 Base Address Bits 32:43: Bits 1:43 can be masked to define an address space from 2^{20} (1 MB) to 2^{63} in size.	
12:31		Reserved	

20.20.13 Outbound Memory Region 2 Mask High Register (PEGPLn_OMR2MSKH)

Functionally, the Outbound Memory Region 2 Mask High and Low Registers can be viewed as a single 64-bit register that defines the size of this address region. The size of the Outbound Memory Region 2 is programmable from 2^{20} (1 MB) to 2^{63} bytes by means of a mask vector. For details about the mask vector, see *Outbound Memory Region 1 Mask High Register (PEGPLn_OMR1MSKH)* on page 559.

Figure 20-76. Outbound Memory Region 2 Mask High Register (PEGPLn_OMR2MSKH)			
0		Reserved	
1:31	BAMH	Outbound Memory Region 2 Mask Bits 1:31: Bits 1:43 of the Base Address can be masked to define an address space from 2^{20} (1MB) to 2^{63} bytes in size. The mask defines the bits that are considered by the address decoder, thus determining the size of the region.	

20.20.14 Outbound Memory Region 2 Mask and Validity Low Register (PEGPLn_OMR2MSKL)

Figure 20-77. Outbound Memory Region 2 Mask and Validity Low Register (PEGPLn_OMR2MSKL)			
0:11	BAML	Outbound Memory Region 2 Mask Bits 32:43. See explanation for PEGPLn_OMR2MSKH[BAMH].	
12:30		Reserved	
31	VAL	Specifies whether Memory Region 2 is valid: 0 The PCI Express PLB Slave interface does not respond to transactions that fall into this region.	

20.20.15 Outbound Memory Region 3 Base Address High Register (PEGPLn_OMR3BAH)

Functionally, the Outbound Memory Region 3 Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the third Outbound Memory Region (BAR2 for outbound transactions).

Figure 20-78. Outbound Memory Region 3 Base Address High Register (PEGPLn_OMR3BAH)			
0:31	BAH	Outbound Memory Region 3 base address Bits 0:31	

20.20.16 Outbound Memory Region 3 Base Address Low Register (PEGPLn_OMR3BAL)

Figure 20-79. Outbound Memory Region 3 Base Address Low Register (PEGPLn_OMR3BAL)			
0:24	BAL	Outbound Memory Region 3 Base Address Bits 32:56: Bits 1:56 can be masked to define an address space from 2^7 (128B) to 2^{63} in size.	
25:31		Reserved	

20.20.17 Outbound Memory Region 3 Mask High Register (PEGPLn_OMR3MSKH)

Functionally, the Outbound Memory Region 3 Mask High and Low Registers can be viewed as a single 64-bit register that defines the size of this address region. The size of the Outbound Memory Region 3 is programmable from 2^7 (128 bytes) to 2^{63} bytes by means of a mask vector. For details about the mask vector, see *Outbound Memory Region 1 Mask High Register (PEGPLn_OMR1MSKH)* on page 559.

Figure 20-80. Outbound Memory Region 3 Mask High Register (PEGPLn_OMR3MSKH)			
0		Reserved	
1:31	BAMH	Outbound Memory Region 3 Mask Bits 1:31: Bits 1:56 of the Base Address can be masked to define an address space from 2^7 (128B) to 2^{63} bytes in size. The mask defines the bits that are considered by the address decoder, thus determining the size of the region.	

20.20.18 Outbound Memory Region 3 Mask and Validity Low Register (PEGPLn_OMR3MSKL)

Figure 20-81. Outbound Memory Region 3 Mask and Validity Low Register (PEGPLn_OMR3MSKL)			
0:24	BAML	Outbound Memory Region 3 Mask Bits 32:56. See explanation for PEGPLn_OMR3MSKH[BAMH].	
25:29		Reserved	
30	IO	Specifies whether Region 3 is dedicated for I/O transactions or used as the third Memory Region: 0 Outbound Memory Region transactions. 1 I/O Region transactions.	
31	VAL	Specifies whether Memory Region 3 is valid: 0 The PCI Express PLB Slave interface does not respond to transactions that fall into this region.	

20.20.19 Local Register Region Base Address High Register (PEGPLn_REGBAH)

Functionally, the Local Register Region Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the Local Register Region (BAR5).

Figure 20-82. Local Register Region Base Address High Register (PEGPLn_REGBAH)

0:31	BAH	Local Register Region Base Address Bits 0:31	

20.20.20 Local Register Region Base Address Low Register (PEGPLn_REGBAL)

Figure 20-83. Local Register Region Base Address Low Register (PEGPLn_REGBAL)

0:24	BAL	Local Register Region Base Address Bits 32:56. Bits 1:56 can be masked to define an address space from 128B (2^7) to 32KB (2^{15}) in size.	
25:31		Reserved	

20.20.21 Local Register Region Mask and Validity Register (PEGPLn_REGMSK)

The size of the Local Register Region is programmable from 128 bytes to 32 KB, by means of a mask vector. For details about the mask vector, see *Configuration Region Mask and Validity Register (PEGPLn_CFGMSK)* on page 557.

Reset Value x0000_0000

Figure 20-84. Local Register Region Mask and Validity Register (PEGPLn_REGMSK)

0:16		Reserved	
17:24	BAM	Register Region Base Address Mask: Bits 49:56 of the Local Register Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 128B (2^7) to 32KB (2^{15}): 00000000 - 32KB 10000000 - 16KB 11000000 - 8KB ---------- 11111111 - 128B	
25:30		Reserved	
31	VAL	Specifies whether the register region is valid: 0 The PCI Express PLB Slave interface does not respond to transactions that fall into this region.	

20.20.22 Special Purpose Prefetch Buffer Handler Register (PEGPLn_SPECIAL)

The PCI Express PLB Slave interface implements four special purpose prefetch buffer handlers. For outbound read transactions, each buffer handler manages a 2-KByte prefetch buffer. PLB Masters assigned to these buffer handlers are identified by their respective PLB Master IDs. Only one special purpose prefetch buffer can be assigned to each bus master.

Figure 20-85. Special Purpose Prefetch Buffer Handler Register (PEGPLn_SPECIAL)

0:3	S1ID	ID of the PLB Master assigned to Special Purpose Prefetch Buffer Handler 1	
4	S1V	Special Purpose Prefetch Buffer Handler 1: 0 Disabled. 1 Supports Outbound Read operations.	
5:7		Reserved	
8:11	S2ID	ID of the PLB Master assigned to Special Purpose Prefetch Buffer Handler 2	
12	S2V	Special Purpose Prefetch Buffer Handler 2: 0 Disabled. 1 Supports Outbound Read operations.	
13:15		Reserved	
16:19	S3ID	ID of the PLB Master assigned to Special Purpose Prefetch Buffer Handler 3	
20	S3V	Special Purpose Prefetch Buffer Handler 3: 0 Disabled. 1 Supports Outbound Read operations.	
21:23		Reserved	
24:27	S4ID	ID of the PLB Master assigned to Special Purpose Prefetch Buffer Handler 4	
28	S4V	Special Purpose Prefetch Buffer Handler 4: 0 Disabled. 1 Supports Outbound Read operations.	
29:31		Reserved	

20.20.23 GPL Configuration Register (PEGPLn_CFG)

Figure 20-86. GPL Configuration Register (PEGPLn_CFG)

0	IWPE	Inbound Write Parity Check Enable	
1	ORPE	Outbound Read Parity Check Enable	
2	PLE	Inbound Read Pipeline Enable	
3	MGA	PCI Express PLB Master Guarded Attribute	
4:5	MPRI	PCI Express PLB Master Priority	
6	DMER	Disable Slave MERR Assertion: Setting this bit to 1 suppresses the error assertion (Read and Write Error) if a transaction error is detected by the PCI Express PLB Slave interface.	
7	DTDC	Disable Target Directed Completion (TDC) Assertion: Setting this bit to 1 suppresses the TDC mechanism in the PCI Express PLB Slave interface.	
8:31		Reserved	

20.20.24 Error Status Register (PEGPLn_ESR)

Bits 0:5 of this register are read only, write 1 to clear (R/W1C). Register bits indicate the status when read. A set bit when write indicates that the status event bit can be cleared. Writing a 0 has no effect.

Figure 20-87. Error Status Register (PEGPLn_ESR)

0	SEV	Slave Error Valid Indication: If asserted, PEGPLn_EARH, PEGPLn_EARL, and PEGPLn_EATR contain valid information.	
1	MIRQ	MIRQ Received Indication: MIRQ exception feedback from any PLB Slave to the PCI Express PLB Master interface.	
2	MERR	MERR Received Indication: Transaction error (Read and Write) indication feedback from any PLB Slave during a transaction initiated by the PCI Express PLB Master interface.	
3	TOT	Time-Out Received Indication: Time-Out indication feedback from the PLB Arbiter during a transaction initiated by the PCI Express PLB Master interface.	
4	ORPE	Outbound Read Parity Error.	
5	IWPE	Inbound Write Parity Error.	
6	IE	Interrupt Indication Enable: If set to 1, the PCI Express DCR interrupt is raised if any of the previous bits (0:5) are set.	
7:31		Reserved	

20.20.25 Slave Error Address High Register (PEGPLn_EARH)

Figure 20-88. Slave Error Address High Register (PEGPLn_EARH)

0:31	SEARH	Upper PLB Address provided to the PCI Express PLB Slave interface if an error occurs	

20.20.26 Slave Error Address Low Register (PEGPLn_EARL)

Figure 20-89. Slave Error Address Low Register (PEGPLn_EARL)

0:31	SEARL	Lower PLB Address provided to the PCI Express PLB Slave interface if an error occurs	

20.20.27 Slave Error Attribute Register (PEGPLn_EATR)

Figure 20-90. Slave Error Attribute Register (PEGPLn_EATR)

0	SERNW	Read Not Write Attribute of the PLB Master addressing the PCI Express PLB Slave interface if an error occurs: 0 PCI Express PLB Slave Write operation. 1 PCI Express PLB Slave Read operation.	
1:4	SESIZE	Size Attribute of the PLB Master addressing the PCI Express PLB Slave interface if an error occurs	
5:8	SEMID	ID Attribute of the PLB Master addressing the PCI Express PLB Slave interface if an error occurs	
9:31		Reserved	

20.21 Application-Specific Registers (PEUTL*n*_xxxx)

The Application-Specific Register Block contains registers that are used for application-specific parameter configuration and status. These registers are not part of the PCI Express configuration space and are directly mapped into the PLB bus address space. Each PCI Express port uses a 256-byte PLB address space. Register read/write access is decoded and performed by the PCI Express PLB Slave interface.

The PLB address mechanism is defined within each PCI Express port by the PEGPLn_REGBAH, PEGPLn_REGBAL, and PEGPLn_REGMSK registers (where n is the port number - 0:1). These are dedicated DCR registers. See pages 562 through 562 for detailed descriptions.

Table 20-16 shows the PLB address offset mapping for each PCI Express port. The letter n in the PEUTLn prefix of each register name represents either 0 or 1, depending on the PCI Express port being addressed.

Table 20-16. Application-Specific PLB Address Mapping

Mnemonic	Register	Address	Access	Page
PEUTLn_PBCTL	PLB Bus Control Register	0x00	R/W	567
PEUTLn_STA	UTL Status Register	0x04	R	568
PEUTLn_PBASTA	PLB Bus Agent Status Register	0x08	R/W1C	568
PEUTLn_PBAEEN	PLB Bus Agent Error Enable Register	0x0C	R/W	569
PEUTLn_PBAIEN	PLB Bus Agent Interrupt Enable Register	0x10	R/W	569
PEUTLn_PBBSZ	PLB Bus Burst Size Configuration Register	0x20	R/W	570
PEUTLn_RID	Revision ID Register	0x24	R	570
PEUTLn_OPHBSZ	Outbound Posted Header Buffer Allocation Register	0x60	R	570
PEUTLn_OPDBSZ	Outbound Posted Data Buffer Allocation Register	0x68	R/W	571
PEUTLn_IPHBSZ	Inbound Posted Header Buffer Allocation Register	0x70	R/W	571
PEUTLn_IPDBSZ	Inbound Posted Data Buffer Allocation Register	0x78	R/W	572
PEUTLn_ONHBSZ	Outbound Non Posted Header Buffer Allocation Register	0x80	R	572
PEUTLn_INHBSZ	Inbound Non Posted Header Buffer Allocation Register	0x88	R/W	573
PEUTLn_OUTTR	Outbound Read Tag Allocation Register	0x90	R/W	574
PEUTLn_INTR	Inbound Read Tag Allocation Register	0x98	R/W	575
PEUTLn_PCTL	PCI Express Port Control Register	0xA0	R/W	575
PEUTLn_PSTA	PCI Express Port Status Register	0xA4	R/W1C	576
PEUTLn_PERREN	PCI Express Port Error Enable Register	0xA8	R/W	577
PEUTLn_PIRQEN	PCI Express Port Interrupt Enable Register	0xAC	R/W	578
PEUTLn_RCSTA	Root Complex Status Register	0xB0	R/W1C	578
PEUTLn_RCERREN	Root Complex Error Enable Register	0xB4	R/W	579
PEUTLn_RCIRQEN	Root Complex Interrupt Enable Register	0xB8	R/W	580
PEUTLn_EPSTA	Endpoint Status Register	0xBC	R/W1C	581
PEUTLn_EPERREN	Endpoint Error Enable Register	0xC0	R/W	582
PEUTLn_EPIRQEN	Endpoint Interrupt Enable Register	0xC4	R/W	582
PEUTLn_PMCTL	Power Management Control Register	0xC8	R/W	583
PEUTLn_PID	PCI Express Port ID Register	0xCC	R	584

20.21.1 PLB Bus Control Register (PEUTLn_PBCTL)

Reset Value x0800_0000

Figure 20-91. PLB Bus Control Register (PEUTLn_PBCTL)

0:3		Reserved	
4	PMTE	PLB Master Enable: 0 Inbound transactions are not presented to the PLB Master, but are accumulated in the port buffers. 1 Inbound transaction are transmitted to the PLB Bus.	R/W
5	PSTD	PLB Slave Transaction Disable: 0 PLB slave transaction disable 1 The UTL prevents PLB slave from issuing outbound requests by de-asserting buffer availability indicators for all outbound reads and writes. This bit must be programmed in conjunction with the SWD0, SWD8, SWD9, SRD0, and SRD8 bits in this register.	R/W
6		Reserved	
7	RDMD	Outbound Read Data Mode Disable: 0 (default) Outbound reads are performed using consecutive data access. 1 Forces random access mode for outbound read data mode.	R/W
8	SWD0	Outbound Memory Write Transaction Disable: 1 The UTL prevents a PLB Master from issuing PCI Express outbound write requests (memory writes) by de-asserting the buffer availability indication for this channel.	R/W
9:15		Reserved	
16	SWD8	Outbound I/O and Configuration Write Transaction Disable: 1 The UTL prevents a PLB Master from issuing PCI Express outbound write requests (I/O, configuration writes) by de-asserting the buffer availability indication for this channel.	R/W
17	SWD9	Outbound Message Write Transaction Disable: 1 The UTL prevents a PLB Master from issuing PCI Express outbound write requests (messages) by de-asserting the buffer availability indication for this channel.	R/W
18	SRD0	Outbound Memory, TC0 Read Transaction Disable: 1 The UTL prevents a PLB Master from issuing PCI Express outbound read requests (memory reads, TC0) by de-asserting the buffer availability indication for this channel.	R/W
19:25		Reserved	
26	SRD8	Outbound I/O and Configuration Read Transaction Disable: 1 The UTL prevents a PLB Master from issuing PCI Express outbound read requests (I/O, configuration reads) by de-asserting the buffer availability indication for this channel.	R/W
27		Reserved Always 0.	R/W
28	CRRE	CRS Resend Enable: 0 Transactions completed with CRS status are reported to GBIF with error and retry indications. 1 Enable reissue of configuration transactions completed with CRS status.	R/W
29:31	CRMC	CRS Resend Limit: Number of attempts to resend the configuration transaction before it completes with an error on GBIF because of a repeating CRS error. 000 2 attempts 001 4 attempts 010 8 attempts 011 Reserved 100 Infinite All other values reserved.	R/W

20.21.2 UTL Status Register (PEUTLn_STA)

Reset Value x0000_0000

Figure 20-92. UTL Status Register (PEUTLn_STA)

0	PBS	PLB Bus Event Status: Set to 1 if any of the PLB Bus Agent Status Register bits are set (see *PLB Bus Agent Status Register (PEUTLn_PBASTA)* on page 568).	
1	PES	PCI Express Port Event Status: Set to 1 if one of the PCI Express Port Status Register bits is set (see *PCI Express Port Status Register (PEUTLn_PSTA)* on page 576).	
2	RCES	Root Complex Event Status: Set to 1 if one of the Root Complex Status Register bits is set (see *Root Complex Status Register (PEUTLn_RCSTA)* on page 578). This bit is implemented for Root Complex configuration, but is reserved for Endpoint configuration, for which read access always returns zero.	
3	EPES	Endpoint Event Status: Set to 1 if one of the Endpoint Status Register bits is set (see *Endpoint Status Register (PEUTLn_EPSTA)* on page 581). This bit is implemented for Endpoint configuration, but is reserved for Root Complex configuration, for which read access always returns zero.	
4:31		Reserved	

20.21.3 PLB Bus Agent Status Register (PEUTLn_PBASTA)

If an error occurs, the relevant bit is set by the hardware. Software must write a 1 to reset the error indication. Writing a 0 has no effect

Reset Value x0000_0000

Figure 20-93. PLB Bus Agent Status Register (PEUTLn_PBASTA)

0	RNPP	Inbound Non Posted Request FIFO (RXNP) Parity Error: 1 Indicates that an RXNP parity error was detected on the array's read port.	
1	RFTP	Inbound Read Tag FIFO (RXFT) Parity Error: 1 Indicates that an RXFT parity error was detected on the array's read port.	
2	RPCP	Inbound Posted Request FIFO (RXPC) Parity Error: 1 Indicates that an RXPC parity error was detected on the array's read port.	
3		Reserved	
4	RCIP	Received Completion Indication FIFO (RXCIF) Parity Error: 1 Indicates that an RXCIF parity error was detected on the array's read port.	
5	RCCP	Received Completion Processing FIFO (RCC) Parity Error: 1 Indicates that an RCC parity error was detected on the array's read port.	
6:31		Reserved	

20.21.4 PLB Bus Agent Error Enable Register (PEUTLn_PBAEEN)

Each bit in this register enables detection of the event indicated by the corresponding bit in the PLB Bus Agent Status Register (see *PLB Bus Agent Status Register (PEUTLn_PBASTA)* on page 568).

Reset Value xF8C0_0000

Figure 20-94. PLB Bus Agent Error Enable Register (PEUTLn_PBAEEN)

0	RNPE	Inbound Non Posted Request FIFO (RXNP) Parity Error Enable	
1	RFTE	Inbound Read Tag FIFO (RXFT) Parity Error Enable	
2	RPCE	Inbound Posted Request FIFO (RXPC) Parity Error Enable	
3		Reserved	
4	RCIE	Received Completion Indication FIFO (RXCIF) Parity Error Enable	
5	RCCE	Received Completion Processing FIFO (RCC) Parity Error Enable	
6:31		Reserved	

20.21.5 PLB Bus Agent Interrupt Enable Register (PEUTLn_PBAIEN)

Each bit in this register enables PCI Express interrupt assertion upon detection of the event indicated by the corresponding bit in the PLB Bus Agent Status Register (see *PLB Bus Agent Status Register (PEUTLn_PBASTA)* on page 568).

Reset Value x0000_0000

Figure 20-95. PLB Bus Agent Interrupt Enable Register (PEUTLn_PBAIEN)

0	RNPI	Inbound Non Posted Request FIFO (RXNP) Parity Error Enable	
1	RFTI	Inbound Read Tag FIFO (RXFT) Parity Interrupt Enable	
2	RPCI	Inbound Posted Request FIFO (RXPC) Parity Interrupt Enable	
3		Reserved	
4	RCII	Received Completion Indication FIFO (RXCIF) Parity Interrupt Enable	
5	RCCI	Received Completion Processing FIFO (RCC) Parity Interrupt Enable	
6:31		Reserved	

20.21.6 PLB Bus Burst Size Configuration Register (PEUTLn_PBBSZ)

This register must not be changed after the link is initialized.

Reset Value x5500_0000

Figure 20-96. PLB Bus Burst Size Configuration Register (PEUTLn_PBBSZ)

0		Reserved	
1:3	PBRS	Maximum PLB Bus Outbound Read Request Size: 000 128 Bytes 001 256 Bytes 010 512 Bytes 011 1,024 Bytes 100 2,048 Bytes All other values are reserved.	
4		Reserved	
5:7	PBWS	Maximum PLB Bus Outbound Write Burst Size: 000 128 Bytes 001 256 Bytes 010 512 Bytes (Default) All other values are reserved.	
8:31		Reserved	

20.21.7 Revision ID Register (PEUTLn_RID)

This register specifies the Revision ID Number and the Branch Revision Number of the PCI Express port.

Reset Value x0020_1000

Figure 20-97. Revision ID Register (PEUTLn_RID)

0:11	RVID	Revision ID Number	
12:19	BRVN	Branch Revision Number	
20:31		Reserved	

20.21.8 Outbound Posted Header Buffer Allocation Register (PEUTLn_OPHBSZ)

This register specifies the outbound posted header buffer allocation. The value is determined statically for each application and cannot be configured by software before the PCI Express port is initialized.

Reset Value 0x0200_0000,
0x0400_0000, or
0x0800_0000 depending on
SDR0_PEn_UTLSET1[OPHDN]

Figure 20-98. Outbound Posted Header Buffer Allocation Register (PEUTLn_OPHBSZ)			
0:3		Reserved	
4:7	OPH	Outbound Posted Header Buffer Size (in header entries): 0000 0 headers 0001 1 header 0010 2 headers 0011 3 headers 0100 4 headers 0101 5 headers 0110 6 headers 0100 7 headers 1000 8 headers All other values are reserved.	R/O
8:31		Reserved	

20.21.9 Outbound Posted Data Buffer Allocation Register (PEUTLn_OPDBSZ)

This register specifies the outbound posted data buffer allocation. The buffer allocation must be less than or equal to the maximum buffer size specified by SDR0_PEn_UTLSET1[OPBFS].

An application is allowed to modify this register only if there are no pending outbound TLPs in the system. Modifying this register when outbound TLPs are awaiting transmission may cause unrecoverable outbound TLP corruption.

Reset Value 0x0000_0000

Figure 20-99. Outbound Posted Data Buffer Allocation Register (PEUTLn_OPDBSZ)			
0		Reserved	
1:7	OPD	Outbound Posted Data Buffer size (in 128-byte units): All values from 0 to 2KB are in increments of 128 bytes, for example, 0000_0100 - 0.5 KB 0000_1000 - 1 KB 0001_0000 - 2 KB All other values above 2KB are reserved.	R/W
8:31		Reserved	

20.21.10 Inbound Posted Header Buffer Allocation Register (PEUTLn_IPHBSZ)

This register specifies the inbound posted header buffer allocation. The buffer allocation must be less than or equal to the maximum buffer size specified by SDR0_PEn_UTLSET1[IPHDN].

Reset Value 0x0400_0000

Figure 20-100. Inbound Posted Header Buffer Allocation Register (PEUTLn_IPHBSZ)			
0:3		Reserved	
4:7	IPH	Inbound Posted Header Buffer Size (in header entries): 0001 1 header 0010 2 headers 0011 3 headers 0100 4 headers 0101 5 headers 0110 6 headers 0100 7 headers 1000 8 headers All other values are reserved.	R/W
8:31		Reserved	

20.21.11 Inbound Posted Data Buffer Allocation Register (PEUTLn_IPDBSZ)

This register specifies the inbound posted data buffer allocation. The buffer allocation must be less than or equal to the maximum buffer size specified by SDR0_PEn_UTLSET1[IPBFS].

Reset Value 0x1000_0000

Figure 20-101. Inbound Posted Data Buffer Allocation Register (PEUTLn_IPDBSZ)			
0:7	IPD	Inbound Posted Data Buffer Size All values from 0 to 2KB are in increments of 128 bytes, for example: 0000_0100 - 0.5 KB 0000_1000 - 1 KB 0001_0000 - 2 KB All other values above 2KB are reserved.	R/W
8:31		Reserved	

20.21.12 Outbound Non Posted Header Buffer Allocation Register (PEUTLn_ONHBSZ)

This register specifies the non posted header buffer allocation. This value is determined statically for each application and cannot be configured by software.

Reset Value x0200_0000, x0400_0000, or x0800_0000 depending on SDR0_PEn_UTLSET1[ONHDN]

Figure 20-102. Outbound Non Posted Header Buffer Allocation Register (PEUTLn_ONHBSZ)			
0:3		Reserved, always zero.	

4:7	ONH	Outbound Non Posted Header Buffer Size (in header entries): 0000 0 headers 0001 1 header 0010 2 headers 0011 3 headers 0100 4 headers 0101 5 headers 0110 6 headers 0100 7 headers 1000 8 headers All other values are reserved	RO
8:31		Reserved	

20.21.13 Inbound Non Posted Header Buffer Allocation Register (PEUTLn_INHBSZ)

This register specifies the inbound non posted header buffer allocation. The buffer allocation must be less than or equal to the size defined by SDR0_PEn_UTLSET1[INHDN].

The reset value of the INH field represents the maximum supported size of the inbound non posted header buffer.

Reset Value: x0200_0000, x0400_0000, or x0800_0000 depending on SDR0_PEn_UTLSET1[INHDN]

Figure 20-103. Inbound Non Posted Header Buffer Allocation Register (PEUTLn_INHBSZ)

0:3		Reserved	
4:7	INH	Inbound Non Posted Header Buffer Size (in header entries): 0001 1 header 0010 2 headers 0011 3 headers 0100 4 headers 0101 5 headers 0110 6 headers 0100 7 headers 1000 8 headers All other values are reserved.	R/W
8:11		Reserved	

20.21.14 Outbound Read Tag Allocation Register (PEUTLn_OUTTR)

This register specifies the outbound read tag buffer allocation. The number of outbound non posted requests allocated must be less than or equal to the number of supported outbound transactions outstanding on the PCI Express interface specified by SDR0_PEn_UTLSET2[ONTRN].

This register can only be programmed when there is no outbound read transaction in the system.

To program a new Outbound Read Tag Allocation value, software must follow this procedure:
1. Poll bit PEUTLn_PCTL[PTAE] until it is '1'. This ensures that no outbound read transaction is pending.
2. Program the new value in PEUTLn_OUTTR.
3. Set bit PEUTLn_PCTL[PTR] = '1' to finish programming the new Tag Allocation parameter.

Reset Value: x0400_0000, x0800_0000, x1000_0000, or x2000_0000 depending on SDR0_PEn_UTLSET2[ONTRN]

Figure 20-104. Outbound Read Tag Allocation Register (PEUTLn_OUTTR)

0:1		Reserved	
2:7	OCT	Number of Outbound Non Posted Requests Outstanding on the PCI Express Interface (in header entries): 0000 0 headers 0001 1 header 0010 2 headers 0011 3 headers 0100 4 headers 0101 5 headers 0110 6 headers 0100 7 headers 1000 8 headers All other values are reserved.	R/W
8:31		Reserved	

20.21.15 Inbound Read Tag Allocation Register (PEUTLn_INTR)

This register specifies the inbound read tag buffer allocation. The number of inbound non posted requests allocated must be less than or equal to the maximum number of inbound non posted transactions outstanding on the PLB bus specified by SDR0_PEn_UTLSET1[INTRN].

At least one inbound read tag must be allocated. This register can only be programmed when there is no inbound read transaction in the system.

To program a new Inbound Read Tag Allocation value, software must follow this procedure:
1. Poll bit PEUTLn_PCTL[GTAE] until it is '1'. This ensures that no inbound read transaction is pending.
2. Program the new value in PEUTLn_INTR.
3. Set bit PEUTLn_PCTL[GTR] = '1' to finish programming the new Tag Allocation parameter.

Reset Value: x0400_0000, x0800_0000, x1000_0000, or x2000_0000 depending on SDR0_PEn_UTLSET1[INTRN]

Figure 20-105. Inbound Read Tag Allocation Register (PEUTLn_INTR)

0:1		Reserved	
2:7	ICT	Number of Inbound Reads Outstanding on the PLB bus (in header entries): 00_00011 header 00_00102 headers 00_00113 headers 00_01004 headers 00_01015 headers 00_01106 headers 00_01007 headers 00_10008 headers All other values are reserved.	R/W
8:31		Reserved	

20.21.16 PCI Express Port Control Register (PEUTLn_PCTL)

Reset Value: x0000_0066

Figure 20-106. PCI Express Port Control Register (PEUTLn_PCTL)

0	VRB	Receive Buffers Configured: 1 Indicates that the application has configured all the required buffers and the PCI Express port is permitted to advertise the configured buffer space during credits initialization upon Transaction Layer request.	
1:7		Reserved	
8	TXE	Transmit Enable: 1 Enables the transmission of outbound PCI Express TLPs to the Transaction Layer.	
9:12		Reserved	

13	PTR	PCI Express Tags Allocation Reset: 0 Has no effect; reads always returns 0. 1 Resets PCI outbound read tags allocation.	RS
14:16		Reserved	
17	PTAE	PCI Express Tags Allocation Reset Enable: 1 No outstanding outbound reads on PCI Express and tags allocation can be reprogrammed.	RO
18		Reserved	
19	GTR	GBIF Tags Allocation Reset: Resets inbound GBIF read tags allocation. 0 Has no effect; reads always returns 0. 1 Triggers tags reset.	RS
20:22		Reserved	
23	GTAE	GBIF Tags Allocation Reset Enable: 1 No outstanding inbound reads on the system bus and tags allocation can be reprogrammed.	RO
24:27	RTOS	Outbound Non Posted Transactions Timeout Configuration: Specifies the transaction timeout value for outbound MemRd, IORd, and IOWr transactions. 0000 125 µs 0001 250 µs 0010 0.5 ms 0011 1 ms 0100 2 ms 0101 4 ms 0110 8 ms (default) 0111 16 ms All other values are reserved.	
28:31	CTOS	Outbound Configuration Transaction Timeout: Specifies the transaction timeout value for outbound CfgRd and CfgWr transactions. 0000 125 µs 0001 250 µs 0010 0.5 ms 0011 1 ms 0100 2 ms 0101 4 ms 0110 8 ms (default) 0111 16 ms 1000 32 ms 1001 64 ms 1010 128 ms 1011 256 ms 1100 512 ms 1101 1 s These bits are implemented for Root Complex configuration, but are reserved for Endpoint configuration, for which read access always returns zero.	

20.21.17 PCI Express Port Status Register (PEUTLn_PSTA)

If an error occurs, the relevant bit is set by the hardware. Software must write a 1 to reset the error indication. Writing a 0 has no effect.

Reset Value 0x0000_0000

Figure 20-107. PCI Express Port Status Register (PEUTLn_PSTA)			
0	TXDP	Outbound Uncorrectable Data Parity Error: 1 Indicates that an uncorrectable data parity error was detected by the Transaction Layer during outbound TLP transmission.	
1	TPCP	Outbound Posted Requests FIFO (TXPC) Parity Error: 1 Indicates that a TXCP parity error was detected on the array's read port.	
2	TNPP	Outbound Non Posted Requests FIFO (TXNP) Parity Error: 1 Indicates that a TXNP parity error was detected on the array's read port.	
3	TFTP	Outbound Non Posted Requests Free Tags FIFO (TXFT) Parity Error: 1 Indicates that a TXFT parity error was detected on the array's read port.	
4	TCAP	Transmit Completions Attribute Array (TXCA) Parity Error: 1 Indicates that a TXCA parity error was detected on the array's read port.	
5	TCIP	Transmit Completions Indications FIFO (TXCIF) Parity Error: 1 Indicates that a TXCIF parity error was detected on the array's read port.	
6	ITHP	Inbound TLP Header Parity Error: 1 Indicates that a parity error was detected in the received TLP header.	
7		Reserved	
8	PLUP	PCI Express LinkUp Status: 1 PCI Express link changed to the LinkUp state.	
9	PLDN	PCI Express LinkDown Status: 1 PCI Express link changed to the LinkDown state.	
10	OTDS	Outbound Request Discard Status: 1 An outbound request TLP was discarded.	
11		Reserved	
12	CIR	Receive Credit Initialization Error: 1 The TL function attempted to initialize credits, but it is not enabled by the PCI Express Control Register.	
13:31		Reserved	

20.21.18 PCI Express Port Error Enable Register (PEUTLn_PERREN)

Each bit in this register enables detection of the event indicated by the corresponding bit in PEUTL*n*_PSTA (see *PCI Express Port Status Register (PEUTLn_PSTA)* on page 576).

Reset Value 0xFEEF_0000

Figure 20-108. PCI Express Port Error Enable Register (PEUTLn_PERREN)			
0	TXDE	Outbound Uncorrectable Data Parity Error Enable	
1	TPCE	Outbound Posted Requests FIFO (TXPC) Parity Error Enable	
2	TNPE	Outbound Non Posted Requests FIFO (TXNP) Parity Error Enable	
3	TFTE	Outbound Non Posted Requests Free Tags FIFO (TXFT) Parity Error Enable	
4	TCAE	Transmit Completion Attribute Array (TXCA) Parity Error Enable	
5	TCIE	Transmit Completions Indications FIFO (TXCIF) Parity Error Enable	

6	ITHE	Inbound TLP Header Parity Error Enable	
7		Reserved, always zero.	
8	PLUE	PCI Express LinkUp Event Enable	
9	PLDE	PCI Express LinkDown Event Enable	
10	OTDE	Outbound Request Discard Event Enable	
11		Reserved, always zero.	
12	CIE	Receive Credit Initialization Error Enable	
13:31		Reserved, always zero.	

20.21.19 PCI Express Port Interrupt Enable Register (PEUTLn_PIRQEN)

Each bit in this register enables PCI Express port interrupt assertion upon detection of the event indicated by the corresponding bit in PEUTL*n*_PSTA (see *PCI Express Port Status Register (PEUTLn_PSTA)* on page 576).

Reset Value 0x0000_0000

Figure 20-109. PCI Express Port Interrupt Enable Register (PEUTLn_PIRQEN)

0	TXDI	Outbound Uncorrectable Data Parity Error Interrupt Enable	
1	TPCI	Outbound Posted Requests FIFO (TXPC) Parity Interrupt Enable	
2	TNPI	Outbound Non Posted Requests FIFO (TXNP) Parity Interrupt Enable	
3	TFTI	Outbound Non Posted Requests Free Tags FIFO (TXFT) Parity Interrupt Enable	
4	TCAI	Transmit Completion Attributes Array (TXCA) Parity Interrupt Enable	
5	TCII	Transmit Completion Indications FIFO (TXCIF) Parity Interrupt Enable	
6	ITHI	Inbound TLP Header Parity Interrupt Enable	
7		Reserved	
8	PLUE	PCI Express LinkUp Interrupt Enable	
9	PLDE	PCI Express LinkDown Interrupt Enable	
10	OTDE	Outbound Request Discard Interrupt Enable	
11		Reserved, always zero	
12	CIE	Receive Credit Initialization Error Interrupt Enable	
13:31		Reserved, always zero	

20.21.20 Root Complex Status Register (PEUTLn_RCSTA)

If an event occurs, the relevant bit is set by the hardware. Software must write a 1 to reset the error indication. Writing a 0 has no effect.

INT*x* legacy interrupt status bits are read only. They are set upon Assert_INTx message reception and reset by the corresponding Deassert_INT*x* message.

This register is implemented for Root Complex configuration, but is reserved for Endpoint configuration, for which read access always returns zero.

Reset Value 0x0000_0000

Figure 20-110. Root Complex Status Register (PEUTLn_RCSTA)

0	RLCS	Local Correctable Error Status: 1 A correctable system error occurred in the Root Port device.	
1	RLNS	Local Non Fatal Error Status: 1 A non fatal system error occurred in the Root Port device.	
2	RLFS	Local Fatal Error Status: 1 A fatal system error occurred in the Root Port device.	
3	RRCS	Remote Correctable Error Status: 1 A correctable system error occurred in the remote PCI Express device.	
4	RRNS	Remote Non Fatal Error Status: 1 A non fatal system error occurred in the remote PCI Express device.	
5	RRFS	Remote Fatal Error Status: 1 A fatal system error occurred in the remote PCI Express device.	
6	RCPS	PME Status: 1 The remote PCI Express device signaled a Power Management Event.	
7	RCTS	PME Turn Off Acknowledge Status: 1 A PME_TO_Ack message was received.	
8	PAAS	INTA Status: 1 Legacy INTA interrupt is asserted.	
9	PABS	INTB Status: 1 Legacy INTB interrupt is asserted.	
10	PACS	INTC Status: 1 Legacy INTC interrupt is asserted.	
11	PADS	INTD Status: 1 Legacy INTD interrupt is asserted.	
12	ALES	ASPM L1Enter Request Status: Set to 1 if a PM_Active_State_Request_L1 DLLP is received.	
13	CRSS	CRS Status: Set to 1 if a configuration request is completed by the destination with Configuration Retry Status (CRS).	
14:31		Reserved	

20.21.21 Root Complex Error Enable Register (PEUTLn_RCERREN)

Each bit in this register, except for the COSE, UNSE, UFSE, and BCSE bits, enables the reporting of the event indicated by the corresponding bit in the PEUTLn_RCSTA register. These four bits reflect PCI configuration space registers and do not affect the PEUTLn_RCSTA register bits or interrupt line assertion.

This register is implemented for Root Complex configuration, but is reserved for Endpoint configuration, for which read access always returns zero.

Reset Value xFFFC_0000

Figure 20-111. Root Complex Error Enable Register (PEUTLn_RCERREN)			
0	RLCE	Local Correctable Error Report Enable	
1	RLNE	Local Non Fatal Error Report Enable	
2	RLFE	Local Fatal Error Report Enable	
3	RRCE	Remote Correctable Error Report Enable	
4	RRNE	Remote Non Fatal Error Report Enable	
5	RRFE	Remote Fatal Error Report Enable	
6	RCPE	PME Report Enable	
7	RCTE	PME Turn Off Acknowledge Report Enable	
8	PAAE	INTA Report Enable	
9	PABE	INTB Report Enable	
10	PACE	INTC Report Enable	
11	PADE	INTD Report Enable	
12	ALEE	ASPM_L1_Enter event Report Enable	
13	CRSE	CRS Report Enable	
14:15		Reserved	
16	COSE	System Error on Correctable Error Enable: Reflection of PECFGn_ECRTCTL[CERRE].	
17	UNSE	System Error on Uncorrectable Non Fatal Error Enable: System Error on Uncorrectable Non Fatal Error Enable: Reflection of PECFGn_ECRTCTL[NERRE].	
18	UFSE	System Error on Uncorrectable Fatal Error Enable: System Error on Uncorrectable Fatal Error Enable: Reflection of PECFGn_ECRTCTL[PERRE].	
19	BCSE	System Error Reporting from the Remote Devices Enable: System Error Reporting from the Remote Devices Enable: Reflection of PECFGn_BCR[SERRE].	
20:31		Reserved	

20.21.22 Root Complex Interrupt Enable Register (PEUTLn_RCIRQEN)

Each bit in this register enables PCI Express port interrupt assertion upon detection of the event indicated by the corresponding bit in the PEUTLn_RCSTA register.

This register is implemented for Root Complex configuration, but is reserved for Endpoint configuration, for which read access always returns zero.

Reset Value x0000_0000

Figure 20-112. Root Complex Interrupt Enable Register (PEUTLn_RCIRQEN)			
0	RLCI	Local Correctable Error Interrupt Enable	
1	RLNI	Local Non Fatal Error Interrupt Enable	
2	RLFI	Local Fatal Error Interrupt Enable	
3	RRCE	Remote Correctable Error Report Enable	
4	RRNE	Remote Non Fatal Error Report Enable	
5	RRFE	Remote Fatal Error Report Enable	
6	RCPI	PME Interrupt Enable	
7	RCTI	PME Turn Off Acknowledge Interrupt Enable	
8	PAAI	INTA Interrupt Enable	
9	PABI	INTB Interrupt Enable	
10	PACI	INTC Interrupt Enable	
11	PADI	INTD Interrupt Enable	
12	ALEI	ASPM_L1_Enter Event Interrupt Enable	
13	CRSI	CRS Interrupt Enable	
14:31		Reserved	

20.21.23 Endpoint Status Register (PEUTLn_EPSTA)

If an error occurs, the relevant bit is set by the hardware. Software must write a 1 to reset the error indication. Writing a 0 has no effect.

This register is implemented for Endpoint configuration, but is reserved for Root Complex configuration, for which read access always returns zero.

Reset Value x0000_0000

Figure 20-113. Endpoint Status Register (PEUTLn_EPSTA)			
0	PTOM	PME_Turn_Off Message Received	
1	PMCS	PM State Changed Event: Set to 1 if PECFGn_PMCSR[STATE] changes. **Note:** This event is indicated only after transmission of completion TLP for the inbound configuration write request that caused the change.	
2	AENS	ASPM L1 Negotiation Abort: Asserted if a PM_Active_State_Nak message is received from the upstream device.	
3:31		Reserved	

20.21.24 Endpoint Error Enable Register (PEUTLn_EPERREN)

The PTOE, PMCE, and AENE bits enable detection of the event indicated by the corresponding bit in PEUTLn_EPSTA. The PMEE bit reflects PECFGn_PMCSR[PMEE] and does not affect the PEUTLn_EPSTA register bits or interrupt line assertion.

This register is implemented for Endpoint configuration, but is reserved for Root Complex configuration, for which read access always returns zero.

Reset Value xE000_8000

Figure 20-114. Endpoint Error Enable Register (PEUTLn_EPERREN)

0	PTOE	PME_Turn_Off Message Received Event Report Enable	
1	PMCE	PM State Changed Event Report Enable	
2	AENE	ASPM L1 Negotiation Abort Event Report Enable	
3:15		Reserved	
16	PMEE	PME Enable: Reflection of PECFGn_PMCSR[PMEE].	
17:31		Reserved	

20.21.25 Endpoint Interrupt Enable Register (PEUTLn_EPIRQEN)

The PTOI and PMCI bits enable PCI Express port interrupt assertion upon detection of the event indicated by the corresponding bit in PEUTL*n*_EPSTA.

This register is implemented for Endpoint configuration, but is reserved for Root Complex configuration, for which read access always returns zero.

Reset Value x0000_0000

Figure 20-115. Endpoint Interrupt Enable Register (PEUTLn_EPIRQEN)

0	PTOI	PME_Turn_Off Message Received Event Interrupt Enable	
1	PMCI	PM State Changed Event Interrupt Enable	
2	AENI	ASPM L1 Negotiation Abort Event Interrupt Enable	
3:31		Reserved	

20.21.26 Power Management Control Register (PEUTLn_PMCTL)

The TOAK, ALET, ALAK, and ALNK bits indicate the status when read. A cleared bit indicates that the register can be set by writing a 1. Writing a 0 has no effect.

Reset Value x0000_0000

Figure 20-116. Power Management Control Register (PEUTLn_PMCTL)

0	TOAK	Transmit PME_TO_Ack Message: Root Port mode: Reserved Endpoint mode: 1 Triggers PME_TO_Ack message transmission. Cleared by hardware after the PME_TO_Ack message is transmitted.	
1	TTOFF	Transmit PME_Turn_Off Message: Endpoint mode: Reserved Root Port mode: 1 Triggers PME_Turn_Off message transmission. Cleared by hardware after the PME_Turn_Off message is transmitted.	RS
2:3	CPMS	Configured PCI PM State: PCI PM state configured in PECFGn_PMCSR[STATE]: 00 D0 01 D1 10 D2 11 D3hot	
4:5		Reserved	
6:7	OPMS	Operational PCI PM State: Actual PCI PM state of the device: 00 D0 01 D1 10 D2 11 D3hot	
8	L1RS	L1 Power State: 1 Indicates that the link is in the L1 link power state.	
9	L23S	L2/L3 Ready Power State: 1 Indicates that the link is in the L2/L3 Ready link power state.	
10	AL1S	ASPM L1 Power State: 1 Indicates that the link is in the ASPM L1 link power state.	
11		Reserved	
12	ALET	ASPM L1 Enter: Root Port mode: Reserved Endpoint mode: 1 Triggers ASPM L1 entry negotiation. Cleared by hardware when the ASPM L1 state entry negotiation is completed and the link enters ASPM L1 state or negotiation is aborted by the upstream port.	
13	ALAK	ASPM L1 Ack: Root Port mode: 1 Triggers ASPM L1 entry acknowledge (transmission of the PM_Request_Ack DLLP to the downstream port that initiated the negotiation). Cleared by hardware when the ASPM L1 state entry negotiation is completed and the link enters ASPM L1 state. Endpoint mode: Reserved	

14	ALNK	ASPM L1 Nack: Root Port mode: 1 Aborts ASPM L1 entry negotiation by triggering PM_Active_State_Nak message transmission to the downstream port that initiated the power state change. Cleared by hardware after the PM_Active_State_Nak message is transmitted. Endpoint mode: Reserved	
15:31		Reserved	

20.21.27 PCI Express Port ID Register (PEUTLn_PID)

Reset Value x0000_0000

Figure 20-117. PCI Express Port ID Register (PEUTLn_PID)

0:15	PCID	PCI Express Port ID: Root Port configuration: Primary Port ID Endpoint configuration: Destination ID of inbound Type0 configuration write request.	
16:31		Reserved	

20.22 PCI Express Legacy Configuration Registers (PECFGn_xxxx)

The registers in the PCI Express configuration space structures support PCI Express implementation types. They support the industry standard PCI Express specifications and functionality for all single-function PCI Express devices—from Root Complex to Endpoints.

The following PCI Express configuration space structures are supported:

- PCI 2.3 Type0 and Type1 Legacy Headers.
- Power Management Capability Structure.
- Message Signaled Interrupt Capability Structure.
- PCI Express Capability Structure.
- VPD Capability Structure.
- Advanced Error Reporting Capability Structure.
- Vendor-Specific Extended Capability Structure.

The third register block is mapped within the PLB bus address space and is accessed by software by means of each PCI Express PLB Slave agent. Each PCI Express port uses 4KB of PLB address space. All PCI Express configuration space registers are limited to a maximum of one doubleword (4 bytes) and are not allowed to cross doubleword boundaries. The PLB base address is defined for each PCI Express port by the following dedicated DCR registers:

- PCI Express port PE0: PEGPL0_CFGBAH, PEGPL0_CFGBAL and PEGPL0_CFGMSK registers.
- PCI Express port PE1: PEGPL1_CFGBAH, PEGPL1_CFGBAL and PEGPL1_CFGMSK registers.
- PCI Express port PE2: PEGPL2_CFGBAH, PEGPL2_CFGBAL and PEGPL2_CFGMSK registers.

20.22.1 Root Port or Endpoint Configuration

Some of the configuration registers at the same address offset may have different meanings depending on whether a PCI Express port is configured as a Root Port or as an Endpoint. For this reason, two mnemonics are used, as shown in *Table 20-18* on page 587.

The Root Port or Endpoint identification is accomplished by means of SDR0_PEn_DLPSET[PTYPE].

SDR0_PEn_DLPSET[PTYPE]	
0000	PCI Express Endpoint
0001	Legacy PCI Express Endpoint
0100	Root Port

20.22.2 PCI Express Configuration Mechanism – Root Complex Applications

If a PCI Express port is a Root Complex, the software-initiated configuration space accesses always originate on the PLB bus. In a Root Complex application, this is the only software path to the configuration space registers because there can never be an upstream configuration space request on the PCI Express link from a downstream device.

In the Root Complex configuration the PCI Express ports implement the software interface to the configuration registers of all the PCI Express devices attached to the PCI Express fabric. Any read/write access to the Root Port PCI Express configuration registers is forwarded to the local PCI Express configuration registers. All configuration transactions addressed to remote PCI Express devices are converted to PCI Express configuration requests (of either type 0 or type 1, depending on the destination bus number) and forwarded by means of the PCI Express link to their final destination.

The PCI Express ports support PCI Express device configuration by means of the memory-mapped configuration mechanism known as enhanced PCI Express configuration mode. The PLB Slave agent of the PCI Express port is responsible for decoding configuration access as well as determining the register destination defined by the address value as specified in *Table 20-17*.

Table 20-17. PCI Express Configuration Address Mapping

System Bus Address	PCI Express Configuration Space
A[35]	Local Configuration Access
A[36:43]	Bus Number[7:0]
A[44:48]	Device Number[4:0]
A[49:51]	Function Number[2:0]
A[52:55]	Extended Register[3:0]
A[56:63]	Register[7:0]

The PCI Express ports do not support multiple concurrent PCI Express configuration transactions. If a configuration transaction is awaiting completion, the PLB Slave agent cannot issue new transactions of this type.

20.22.3 Root Complex PCI Express Configuration Registers Access

The PLB Slave access to the local PCI Express configuration space is identified by the A[35] bit of the address attribute set to 1 and forwarded to the local configuration registers. Data supplied by the configuration registers is returned to the PLB Slave.

20.22.4 Remote PCI Express Devices Configuration Registers Access

All remote PCI Express configuration transactions are identified by the A[35] bit of the address attribute being set to 0. Transactions with a destination bus number that matches the PCI Express secondary bus number are transmitted as PCI Express Type0 configuration transactions. All PCI Express configuration transactions with a destination bus number not matching the PCI Express port secondary bus numbers are transmitted as PCI Express Type1 configuration transactions.

Configuration transactions are converted to PCI Express non posted transactions and require a specific completion indication from the transaction target. The PCI Express port creates a non posted request TLP, which is placed in a non posted buffer and presented to the Transaction Layer (TL) logic for transmission on the outbound PCI Express link. The PLB Slave is notified of a transaction's completion status upon arrival of a configuration completion TLP from the transaction destination.

20.22.5 PCI Express Configuration Mechanism – Endpoint Applications

If a PCI Express port is anything other than a Root Complex (that is, either a PCI Express Switch Port or an Endpoint), all software-initiated configuration space accesses come from the PCI Express link.

In the Endpoint configuration, the PCI Express port forwards inbound Type0 configuration requests to the local configuration space (Type0 header). The Endpoint sets SDR0_PEn_RCSSET[RDY] to 1, which asserts the READY signal, to indicate that it can be accessed by the configuration software. If the READY signal is de-asserted, the Endpoint replies to all inbound Type0 configuration requests with a completion that indicates Configuration Retry Status (CRS).

The Endpoint function captures the extended register number, bus number, and device number from the configuration request header to construct the local device ID, which is used to identify outbound PCI Express transactions. Type1 configuration requests are processed as unsupported requests. The PCI Express port supports only single-function PCI Endpoint devices, thus only function 0 is supported. The function number field in inbound configuration write requests is ignored. The function number field in inbound configuration read requests must be 0.

20.22.6 PLB Address Mapping

The following tables provide the PLB address mapping for each PCI Express port. The character n in the PECFGn prefix of each register name represents 0, 1, or 2, depending on the PCI Express port being addressed.

The PCI Express register mapping for a Root Port differs from that of an Endpoint. *Table 20-18* summarizes the address mapping for both a Root Port and an Endpoint.

20.22.7 Root Port and Endpoint Configuration Registers

Table 20-18 shows the PCI Express configuration space registers for Root Ports and Endpoints.

Table 20-18. Root Port and Endpoint Configuration Registers

Mnemonic	Register	Address	Access	Page
PECFGn_VENDEVID	PCI Device ID and Vendor ID	0x000	R	592
PECFGn_CMDSTATUS	PCI Status and Command	0x004	R/W	593
PECFGn_REVIDCLASS	PCI Class Codes and Revision ID	0x008	R	594
PECFGn_CACHELS	PCI Header Type and Cache Line Size	0x00C	R/W	594
PECFGn_BAR0L	PCI Base Address Register 0 Low	0x010	R/W	595
PECFGn_BAR0H	PCI Base Address Register 0 High	0x014	R/W	595
PECFGn_BAR1 PECFGn_BUSNUM	**Endpoint:** PCI Endpoint Base Address Register 1 **Root Port:** PCI Subordinate Bus Number/Secondary Bus Number/Primary Bus Number	0x018	R/W R/W	596 599
Reserved PECFGn_IOBASE	**Endpoint:** Reserved **Root Port:** PCI Secondary Status, I/O Limit, and I/O Base	0x01C	R/W	599
PECFGn_BAR2L PECFGn_MEMBAS	**Endpoint:** Endpoint Base Address Register 2 Low **Root Port:** PCI Memory Limit and Memory Base	0x020	R/W R/W	597 600
PECFGn_BAR2H PECFGn_FTCHBAS	**Endpoint:** PCI Endpoint Base Address Register 2 High **Root Port:** PCI Prefetchable Limit and Prefetchable Base	0x024	R/W R/W	597 600
Reserved PECFGn_FTCHBUP	**Endpoint:** Reserved **Root Port:** PCI Prefetchable Memory Base High	0x028	R/W	601
PECFGn_SBSYSVID PECFGn_FTCHLUP	**Endpoint:** PCI Subsystem ID and Subsystem Vendor ID **Root Port:** PCI Prefetchable Memory Limit High	0x02C	R R/W	597 601
PECFGn_EROMBA PECFGn_IOBLUP	**Endpoint:** PCI Endpoint Expansion ROM Base Address **Root Port:** PCI I/O Limit High and I/O Base High	0x030	R/W R/W	598 601
PECFGn_CAP	PCI Capability Pointer	0x034	R	595
Reserved PECFGn_RTEROM	**Endpoint:** Reserved **Root Port:** PCI Root Port Expansion ROM Base Address	0x038	R/W	601
PECFGn_INTLNPN PECFGn_BCR	**Endpoint:** PCI Interrupt Line and Interrupt Pin **Root Port:** PCI Bridge Control	0x03C	R/W R/W	595 602
PECFGn_PMCAP	Power Management Capability ID, Next Pointer, and Capability	0x040	R	603
PECFGn_PMCSR	Power Management Status and Control	0x044	R/W	604
PECFGn_OMCAP	MSI Message Control, Next Pointer, and MSI Capability ID	0x048	R/W	604
PECFGn_ OMMAL	MSI Message Lower Address	0x04C	R/W	605
PECFGn_ OMMAU	MSI Message Upper Address	0x050	R/W	605
PECFGn_OMMDATA	MSI Message Data	0x054	R/W	606
PECFGn_ECCAPID	EC PCI Express Capability, Capability ID, and Next Pointer	0x058	R	606
PECFGn_ECDEVCAP	EC Device Capability	0x05C	R	607
PECFGn_ECDEVCTL	EC Device Status and Control Status	0x060	R/W	608
PECFGn_ECLNKCAP	EC Link Capability	0x064	R	609
PECFGn_ECLNKCTL	EC Link Status and Link Control	0x068	R/W	610

Table 20-18. Root Port and Endpoint Configuration Registers (Continued)

Mnemonic	Register	Address	Access	Page
Reserved PECFGn_ECSLTCAP	**Endpoint:** Reserved **Root Port:** EC Slot Capability	0x06C	R R	611
Reserved PECFGn_ECSLTCTL	**Endpoint:** Reserved **Root Port:** EC Status and Slot Control	0x070	R R/W	612
Reserved PECFGn_ECRTCTL	**Endpoint:** Reserved **Root Port:** EC Root Control	0x074	R R/W	613
Reserved PECFGn_CRSCS	**Endpoint:** Reserved **Root Port:** CRS Completion Status	0x076	R R	613
Reserved PECFGn_ECRTSTA	**Endpoint:** Reserved **Root Port:** EC Root Status	0x078	R R/W	614
PECFGn_VPDCAP Reserved	**Endpoint:** VPD Flag, Address, Next Pointer, and Capability ID **Root Port:** Reserved	0x07C	R/W R	614
PECFGn_VPDDATA Reserved	**Endpoint:** VPD Data **Root Port:** Reserved	0x080	R/W R	615
	Undefined	0x084 0x0FC		
PECFGn_AERCAP	AER Next Capability Offset, Capability Version, and PCI Express EC ID	0x100	R	616
PECFGn_AERUESTA	AER Uncorrectable Error Status	0x104	R/W	617
PECFGn_AERUEMSK	AER Uncorrectable Error Mask	0x108	R/W	617
PECFGn_AERUESEV	AER Uncorrectable Error Severity	0x10C	R/W	618
PECFGn_AERCESTA	AER Correctable Error Status	0x110	R/W	619
PECFGn_AERCEMSK	AER Correctable Error Mask	0x114	R/W	619
PECFGn_AERCAPCTL	AER Capability and Control	0x118	R/W	620
PECFGn_AERHLOG1	AER Header Log Register 1	0x11C	R	620
PECFGn_AERHLOG2	AER Header Log Register 2	0x120	R	620
PECFGn_AERHLOG3	AER Header Log Register 3	0x124	R	620
PECFGn_AERHLOG4	AER Header Log Register 4	0x128	R	620
Reserved PECFGn_AERRTCTL	**Endpoint:** Reserved **Root Port:** AER Root Error Control	0x12C	R/W	621
Reserved PECFGn_AERRTSTA	**Endpoint:** Reserved **Root Port:** AER Root Error Status	0x130	R/W	622
Reserved PECFGn_AERERRSID	**Endpoint:** Reserved **Root Port:** AER Error Source Identification	0x134	R	622
	Undefined	0x138 0x1F4		
PECFGn_VSECCAP	VSEC Next Capability Offset, Capability Version, and EC ID	0x1F8	R	623
PECFGn_VSECID	VSEC Length, Revision, and ID	0x1FC	R	623
PECFGn_VENDEVIDPA	PCI Device ID and Vendor ID Programming Access	0x200	R/W	625
	Reserved	0x204		

Table 20-18. Root Port and Endpoint Configuration Registers (Continued)

Mnemonic	Register	Address	Access	Page
PECFGn_REVIDCLASSPA	PCI Class Codes, Revision ID Programming Access	0x208	R/W	625
	Reserved	0x20C		
PECFGn_BAR0LMPA	PCI Base Address Register 0 Low Mask Programming Access	0x210	R/W	626
PECFGn_BAR0HMPA	PCI Base Address Register 0 High Mask Programming Access	0x214	R/W	626
PECFGn_BAR1MPA	PCI Base Address Register 1 Mask Programming Access	0x218	R/W	627
	Reserved	0x21C		
PECFGn_BAR2LMPA	PCI Base Address Register 2 Low Mask Programming Access	0x220	R/W	627
PECFGn_BAR2HMPA	PCI Base Address Register 2 High Mask Programming Access	0x224	R/W	628
	Reserved	0x228		
PECFGn_SBSYSVIDPA Reserved	**Endpoint:** PCI Subsystem ID/Vendor ID Programming Access **Root Port:** Reserved	0x22C	R/W	628
PECFGn_EROMBAPA Reserved	**Endpoint:** PCI Endpoint Expansion ROM Base Address Mask Programming Access **Root Port:** Reserved	0x230	R/W	629
PECFGn_CAPPA	PCI Capability Pointer Programming Access.	0x234	R/W	629
Reserved PECFGn_RTEROMPA	**Endpoint:** Reserved **Root Port:** PCI Root Port Expansion ROM Base Address Mask Programming Access	0x238	R/W	630
PECFGn_INTPPA	PCI Interrupt Pin Programming Access	0x23C	R/W	630
	Reserved	0x240 0x254		
PECFGn_ECCAPIDPA	EC PCI Express Capability Programming Access	0x258	R/W	631
PECFGn_ECDEVCAPPA	EC Device Capability Programming Access	0x25C	R/W	631
	Reserved	0x260		
PECFGn_ECLNKCAPPA	EC Link Capability Programming Access	0x264	R/W	631
PECFGn_ECLNKCTLPA	EC Link Status and Link Control Programming Access	0x268	R/W	632
Reserved PECFGn_ECSLTCAPPA	**Endpoint:** Reserved **Root Port:** EC Slot Capability Programming Access	0x26C	R/W	632
	Reserved	0x270 0x314		
PECFGn_AERCAPCTLPA	AER Capability and Control Programming Access	0x318	R/W	633
	Reserved	0x31C 0x32C		
Reserved PECFGn_AERRTSTAPA	**Endpoint:** Reserved **Root Port:** AER Root Error Status Programming Access	0x330	R/W	633
	Reserved	0x334		
PECFGn_VSECIDPA	VSEC Revision and VSEC ID Programming Access	0x338	R/W	633
PECFGn_PIMEN	InRegion Valid Programming Access	0x33C	R/W	635

Table 20-18. Root Port and Endpoint Configuration Registers (Continued)

Mnemonic	Register	Address	Access	Page
PECFGn_PIM0LAL	PIM0 Address Low for BAR0	0x340	R/W	635
PECFGn_PIM0LAH	PIM0 Address High for BAR0	0x344	R/W	636
PECFGn_PIM1LAL	PIM1 Address Low for BAR0	0x348	R/W	636
PECFGn_PIM1LAH	PIM1 Address High for BAR0	0x34C	R/W	636
PECFGn_PIM01SAL	PIM01 Size Address Low for BAR0	0x350	R/W	637
PECFGn_PIM01SAH	PIM01 Size Address High for BAR0	0x354	R/W	637
PECFGn_PIM2LAL	PIM2 Address Low for BAR1	0x358	R/W	638
PECFGn_PIM2LAH	PIM2 Address High for BAR1	0x35C	R/W	638
PECFGn_PIM3LAL	PIM3 Address Low for BAR2	0x360	R/W	638
PECFGn_PIM3LAH	PIM3 Address High for BAR2	0x364	R/W	639
PECFGn_PIM4LAL	PIM4 Address Low for BAR2	0x368	R/W	639
PECFGn_PIM4LAH	PIM4 Address High for BAR2	0x36C	R/W	639
PECFGn_PIM34SAL	PIM34 Size Address Low for BAR2	0x370	R/W	640
PECFGn_PIM34SAH	PIM34 Size Address High for BAR2	0x374	R/W	640
PECFGn_PIM5LAL	PIM5 Address Low for BAR3	0x378	R/W	641
PECFGn_PIM5LAH	PIM5 Address High for BAR3	0x37C	R/W	641
PECFGn_POM0LAL	POM0 Address Low for BAR0	0x380	R/W	641
PECFGn_POM0LAH	POM0 Address High for BAR0	0x384	R/W	642
PECFGn_POM1LAL	POM1 Address Low for BAR1	0x388	R/W	642
PECFGn_POM1LAH	POM1 Address High for BAR1	0x38C	R/W	642
PECFGn_POM2LAL	POM2 Address Low for BAR2	0x390	R/W	643
PECFGn_POM2LAH	POM2 Address High for BAR2	0x394	R/W	643
PECFGn_POM3LAL	POM3 Address Low for BAR3	0x398	R/W	643
PECFGn_POM3LAH	POM3Address High for BAR3	0x39C	R/W	644
PECFGn_POM4LAL	POM4 Address Low for BAR4	0x3A0	R/W	644
PECFGn_POM4LAH	POM4 Address High for BAR4	0x3A4	R/W	644
	Reserved	0x3A8		
Reserved PECFGn_RTSID	**Endpoint:** Reserved **Root Port:** Root Port Source ID	0x3AC	R/W	645
Reserved PECFGn_RTBAR1	**Endpoint:** Reserved **Root Port:** Root Port BAR1 Base Address	0x3B0	R/W	645
	Reserved	0x3B4		
Reserved PECFGn_RTBAR2L	**Endpoint:** Reserved **Root Port:** Root Port BAR2 Base Address Low	0x3B8	R/W	645
Reserved PECFGn_RTBAR2H	**Endpoint:** Reserved **Root Port:** Root Port BAR2 Base Address High	0x3BC	R/W	646

Table 20-18. Root Port and Endpoint Configuration Registers (Continued)

Mnemonic	Register	Address	Access	Page
	Reserved	0x3C0 0x3FC		
PECFGn_DLLPPA	Data Link Layer Parameters Programming Access	0x400	R/W	646
	Reserved	0x404 0x424		
PECFGn_ADERC	Advisory Error Reporting Control	0x428	R/W	646
	Reserved	0x42C 0xFFC		

20.23 Configuration Space Registers

The following sections describe the following configuration space registers:

- *PCI Legacy Header Configuration Space Registers*
  - *Common PCI Legacy Header Registers* on page 592
  - *Type0-Specific PCI Legacy Header Registers* on page 596
  - *Type1-Specific PCI Legacy Header Registers* on page 598
- *PCI Configuration Space Capability Structures*
  - *PCI Express Power Management (PM) Capability Structure* on page 603
  - *PCI Express Message Signaled Interrupt Capability Structure* on page 604
  - *PCI Express Capability Structure* on page 606
  - *PCI Express VPD Capability Structure* on page 614
- *PCI Express Extended Capabilities Structures*
  - *PCI Express Advanced Error Reporting Extended Capability Structure* on page 616
  - *Vendor-Specific Extended Capability (VSEC) Structure* on page 623
    - *Programming Access Registers* on page 624
    - *VSEC PIM and POM Registers* on page 633

20.24 PCI Legacy Header Configuration Space Registers

The PCI legacy configuration space registers are required for all PCI specification-defined devices, including PCI Express. This area of the PCI Express configuration space is always non-relocatable and must exist in the address map starting at address offset 0x00 and ending at address offset 0x3F. The PCI legacy header is always either Type0 (all Endpoint devices) or Type1 (bridge, Root Port, or switch). The PCI Express CFG function always prefixes these registers with the PCI designator.

The following sections describe the following:

- Registers that are common for both Type0 and Type1 devices
- Type0-Specific registers
- Type1-Specific registers

20.24.1 Common PCI Legacy Header Registers

The registers shown in *Table 20-19* are required for both Type0 (Endpoint) and Type1 (Root Port) PCI devices.

Table 20-19. Common PCI Legacy Header Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_VENDEVID	0x000	R	R	PCI Device ID	592
		R	R	PCI Vendor ID	592
PECFGn_CMDSTATUS	0x004	R/W1C	R/W1C	PCI Status	593
		R/W	R/W	PCI Command	593
PECFGn_REVIDCLASS	0x008	R	R	PCI Class Codes	594
		R	R	PCI Revision ID	594
PECFGn_CACHELS	0x00C	R	R	PCI Header Type	594
		R/W	R/W	PCI Cache Line Size	594
PECFGn_BAR0L	0x010	R/W	R/W	PCI Base Address Register 0 Low	595
PECFGn_BAR0H	0x014	R/W	R/W	PCI Base Address Register 0 High	595
PECFGn_CAP	0x034	R	R	PCI Capability Pointer	595
PECFGn_INTLNPN	0x03C	R/W	R/W	PCI Interrupt Line	595
		R/W	R/W	PCI Interrupt Pin	595

20.24.1.1 PCI Device ID and Vendor ID Registers (PECFGn_VENDEVID)

These registers are used to communicate PCI Express port identification information to the operating system. The Device ID value is selected by the device supplier to identify the supplier-specific device type. The Vendor ID value is selected by the PCI SIG to ensure unique supplier identification.

Figure 20-118. PCI Device ID and Vendor ID Registers (PECFGn_VENDEVID)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16	DEVID	Device ID	Software Initialization See PECFGn_VENDEVIDPA[DID]	R	R
15:0	VENDEVID	Vendor ID	Software Initialization See PECFGn_VENDEVIDPA[VID]	R	R

20.24.1.2 PCI Status and Command Registers (PECFGn_CMDSTATUS)

These registers are used to communicate PCI Express port status and control information between the hardware and software.

Figure 20-119. PCI Status and Command Registers (PECFGn_CMDSTATUS)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31	DEPE	Detected Parity Error: 1 The PCI Express port has received a poisoned TLP.	0b0	R/W1C	R/W1C
30	SSE	Signaled System Error: 1 The PCI Express port sent fatal or non fatal messages and the SERE bit in this register is set to 1.	0b0	R/W1C	R/W1C
29	RMA	Received Master Abort: 1 The PCI Express port has received Completion with Unsupported Request.	0b0	R/W1C	R/W1C
28	RTA	Received Target Abort: 1 The PCI Express port has received Completion Abort.	0b0	R/W1C	R/W1C
27	STA	Signaled Target Abort: 1 The PCI Express port has signaled Completion Abort.	0b0	R/W1C	R/W1C
26:25		Reserved	0b00	R	R
24	DPE	Master Data Parity Error: 1 The PCI Express port has signaled a poisoned TLP.	0b0	R/W1C	R/W1C
23:21		Reserved	0b000	R	R
20	CL	Capability List	0b1	R	R
19	INTS	Interrupt Status: 1 Indicates INTx pending	0b0	R	R
18:11		Reserved	0x00	R	R
10	INTD	Interrupt Disable: Controls INTx messages	0b0	R/W	R/W
9		Reserved	0b0	R	R
8	SERE	System Error (SERR) Enable	0b0	R/W	R/W
7		Reserved	0b0	R	R
6	PER	Parity Error (PERR) Enable	0b0	R/W	R/W
5:3		Reserved	0b000	R	R
2	PME	Bus Master Enable	0b0	R/W	R/W
1	MEMA	Memory Space Enable	0b0	R/W	R/W
0	IOA	I/O Space Enable	0b0	R/W	R/W

20.24.1.3 PCI Class Codes and Revision ID Registers (PECFGn_REVIDCLASS)

These registers are used to communicate additional PCI Express port identification information to the operating system.

The Class Code values identify the generic function of the port- and register-level programming interface.

The Revision ID value is selected by the device supplier and is to be viewed as a vendor-defined extension to the device ID.

Figure 20-120. PCI Class Codes and Revision ID Registers (PECFGn_REVIDCLASS)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24	BASE	Base Class Code	Software Initialization See PECFGn_REVIDCLASSPA[BASE]	R	R
23:16	SUB	Sub-Class Code	Software Initialization See PECFGn_REVIDCLASSPA[SUB]	R	R
15:8	INT	Interface Class Code	Software Initialization See PECFGn_REVIDCLASSPA[INT]	R	R
7:0	RID	Revision ID	Software Initialization See PECFGn_REVIDCLASSPA[RID]	R	R

20.24.1.4 PCI Header Type and Cache Line Size Registers (PECFGn_CACHELS)

This register is used by the software to determine the register memory map of the remaining PCI Legacy header registers.

The PCI Cache Line Size Register is required for backward PCI compatibility, but has no effect on PCI Express port functionality.

Figure 20-121. PCI Header Type and Cache Line Size Registers (PECFGn_CACHELS)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24		Reserved	0x00	R	R
23:16	HDTYPE	Header Type: 0x00 - Endpoint 0x01 - Root Complex	 0x00 0x01	R	R
15:8		Reserved	0x00	R	R
7:0	CACHLS	Cache Line Size	0x00	R/W	R/W

20.24.1.5 PCI Base Address Register 0 Low (PECFGn_BAR0L)

Functionally, the PCI Base Address Register 0 Low and High can be viewed as a single 64-bit register that defines the PCI Express Base Address 0 (BAR0 for inbound transactions).

Figure 20-122. PCI Base Address Register 0 Low (PECFGn_BAR0L)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:4	BAL	PCI Base Address 0 Low (64-bit BAR0)	0x000_0000	R/W	R/W
3	PEF	Prefetchable	Software Initialization See PECFGn_BAR0LMPA[PEF]	R	R
2:1	LT	Memory Type: 64-bit Access Space	Software Initialization See PECFGn_BAR0LMPA[LT]	R	R
0	MSI	Space Indicator: Memory	0b0	R	R

20.24.1.6 PCI Base Address Register 0 High (PECFGn_BAR0H)

Figure 20-123. PCI Base Address Register 0 High (PECFGn_BAR0H)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	BAH	PCI Base Address 0 High (64-bit BAR0)	0x0000_0000	R/W	R/W

20.24.1.7 PCI Capability Pointer Register (PECFGn_CAP)

The PCI Capability Pointer Register contains the linked list pointer to the first PCI capability structure.

Figure 20-124. PCI Capability Pointer Register (PECFGn_CAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:8		Reserved	0x00_0000	R	R
7:0	CAP	Capability Pointer: 0x40 - Power Management Capability 0x7C - Vital Product Data Capability	Software Initialization See PECFGn_CAPPA[CAP]	R	R

20.24.1.8 PCI Interrupt Line and Interrupt Pin Registers (PECFGn_INTLNPN)

These registers are always the first two bytes located at configuration space address 0x03C. However, the upper two bytes of doubleword address 0x03C for Type0 headers are not used the same way as they are for Type1 headers. For Type0 headers, the upper two bytes are reserved. For a description of how the upper two bytes are used for Type1 headers, see *PCI Bridge Control Register (PECFGn_BCR)* on page 602.

The PCI Interrupt Line Register is used by the system software to save the system interrupt routing number associated with the PCI Express port. This register has no effect on PCI device hardware functionality.

The PCI Interrupt Pin Register is used to indicate to the system software whether the PCI Express port has an interrupt pin, and the INT*x* pin to which it is connected.

Figure 20-125. PCI Interrupt Line and Interrupt Pin Registers (PECFGn_INTLNPN)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16		PCI Type0: Reserved PCI Type1: See *PCI Bridge Control Register (PECFGn_BCR)* on page 602.	0x00 0x00	R NA	R R/W
15:8	INTPN	Interrupt Pin: 0x00 - Not connected to INTx 0x01 - Connected to INTA 0x02 - Connected to INTB 0x03 - Connected to INTC 0x04 - Connected to INTD	Software Initialization See PECFGn_INTPPA[INTPN]	R	R
7:0	INTLN	Interrupt Line.	0x00	R/W	R/W

20.24.1.9 Type0-Specific PCI Legacy Header Registers

The following registers are required for all PCI Express Type0 ports (Endpoint).

Table 20-20. Type0-Specific PCI Legacy Header Registers

Register	Address Doubleword Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_BAR1	0x018	R/W	R/W	PCI Endpoint Base Address Register 1	596
PECFGn_BAR2L	0x020	R/W	R/W	PCI Endpoint Base Address Register 2 Low	597
PECFGn_BAR2H	0x024	R/W	R/W	PCI Endpoint Base Address Register 2 High	597
PECFGn_SBSYSVID	0x02C	R	R	PCI Subsystem Vendor ID	597
		R	R	PCI Subsystem ID	597
PECFGn_EROMBA	0x030	R/W	R/W	PCI Endpoint Expansion ROM Base Address	598

20.24.1.10 PCI Endpoint Base Address Register 1 (PECFGn_BAR1)

This register specifies PCI Express Base Address 1 (BAR1 for inbound transactions).

Figure 20-126. PCI Endpoint Base Address Register 1 (PECFGn_BAR1)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:2	BAR1	Endpoint Base Address 1 (32-bit BAR1)	Software Initialization See PECFGn_BAR1MPA[BAR1]	R/W	R/W
1		Reserved	0b0	R	R
0	IO	Space Indicator: I/O	0b1	R	R

20.24.1.11 PCI Endpoint Base Address Register 2 Low (PECFGn_BAR2L)

Functionally, the PCI Endpoint Base Address Registers 2 Low and High registers can be viewed as a single 64-bit register that specifies PCI Express Base Address 2 (BAR2 for inbound transactions).

Figure 20-127. PCI Endpoint Base Address Register 2 Low (PECFGn_BAR2L)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:4	BAL	Endpoint Base Address 2 Low (64-bit BAR2)	0x 000_0000	R/W	R/W
3	PEF	Prefetchable	Software Initialization See PECFGn_BAR2LMPA[PEF]	R	R
2:1	LT	Type: 0 32-bit address memory space (not supported) 1 64-bit address memory space	Software Initialization See PECFGn_BAR2LMPA[BAT]	R	R
0	MSI	Space Indicator: Memory	0b0	R	R

20.24.1.12 PCI Endpoint Base Address Register 2 High (PECFGn_BAR2H)

Figure 20-128. PCI Endpoint Base Address Register 2 High (PECFGn_BAR2H)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	BAH	Endpoint Base Address 2 High (64-bit BAR2)	Software Initialization See PECFGn_BAR2HMPA[BAH]	R/W	R/W

20.24.1.13 PCI Subsystem ID and Subsystem Vendor ID Register (PECFGn_SBSYSVID)

The PCI Subsystem ID value is subsystem vendor-specific. The PCI Subsystem Vendor ID value is allocated by the PCI SIG to ensure unique subsystem supplier identification.

These registers are used to communicate card or subsystem identification information to the operating system. They registers provide a means for card and subsystem vendors to uniquely identify their product separately from the PCI port.

Figure 20-129. PCI Subsystem ID and Subsystem Vendor ID Register (PECFGn_SBSYSVID)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16	SID	Subsystem ID	Software Initialization See PECFGn_SBSYSVIDPA[SID]	R	R
15:0	SVID	Subsystem Vendor ID	Software Initialization See PECFGN_SBSYSVIDPA[SVID]	R	R

20.24.1.14 PCI Endpoint Expansion ROM Base Address Register (PECFGn_EROMBA)

This register is used by devices that need ROM code for port-specific initialization or system-boot functionality. The Expansion ROM BAR is very similar in operation to the standard base address register (BAR).

Both Type0 and Type1 header types have an Expansion ROM Base Address Register. However, this register is located at different offsets in the configuration space address map, depending on the PCI Header Type (Type0=0x030, Type1=0x038). The register definition and operation is the same for both Type0 and Type1 headers, but has been included in both sections of this document because of the different address offsets associated with the two header types.

Figure 20-130. PCI Endpoint Expansion ROM Base Address Register (PECFGn_EROMBA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:11	BAR	Endpoint Expansion ROM Base Address (32-bit BAR3)	0x0_0000	R/W	R/W
10:1		Reserved	0b00_0000_0000	R	R
0	MSI	Expansion ROM Enable 1 Enable (disabled at reset)	Software Initialization See PECFGn_EROMBAPA[ROME]	R	R

20.24.2 Type1-Specific PCI Legacy Header Registers

The following registers are required for all PCI Express Type1 devices (Root Complex, Switch, Bridge).

Table 20-21. Type1-Specific PCI Legacy Header Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_BUSNUM	0x018	N/A	R/W	PCI Primary Bus Number	599
		N/A	R/W	PCI Secondary Bus Number	599
		N/A	R/W	PCI Subordinate Bus Number	599
PECFGn_IOBASE	0x01C	N/A	R/W	PCI I/O Base	599
		N/A	R/W	PCI I/O Limit	599
		N/A	R/W1C	PCI Secondary Status	599
PECFGn_MEMBAS	0x020	N/A	R/W	PCI Memory Base	600
		N/A	R/W	PCI Memory Limit	600
PECFGn_FTCHBAS	0x024	N/A	R/W	PCI Prefetchable Memory Limit	600
		N/A	R/W	PCI Prefetchable Memory Base	600
PECFGn_FTCHBUP	0x028	N/A	R/W	PCI Prefetchable Memory Base High	601
PECFGn_FTCHLUP	0x02C	N/A	R/W	PCI Prefetchable Memory Limit High	601
PECFGn_IOBLUP	0x030	N/A	R/W	PCI I/O Limit High	601
		N/A	R/W	PCI I/O Base High	601
PECFGn_RTEROM	0x038	N/A	R/W	PCI Root Port Expansion ROM Base Address	601
PECFGn_BCR	0x03C	N/A	R/W	PCI Bridge Control	602

PCI Subordinate Bus Number/Secondary Bus Number/Primary Bus Number (PECFGn_BUSNUM)

These registers are provided for software bus number assignment in the PCI bus hierarchy. The hardware uses the value programmed in these registers for decoding and processing PCI configuration space transactions.

Figure 20-131. PCI Subordinate Bus Number/Sec. Bus Number/Pri. Bus Number (PECFGn_BUSNUM)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24		Reserved	0x00	NA	R
23:16	SUBBN	Subordinate Bus Number	0x00	NA	R/W
15:8	SECBN	Secondary Bus Number	0x00	NA	R/W
7:0	PRIBN	Primary Bus Number	0x00	NA	R/W

20.24.2.1 PCI Secondary Status, I/O Limit, and I/O Base Registers (PECFGn_IOBASE)

This register is similar in function and bit definition to the status-related fields in PECFG*n*_CMDSTATUS. However, this register reflects the status of the secondary bus, while PECFG*n*_CMDSTATUS reflects status of the primary bus.

The PCI I/O Limit and I/O Base registers are used to control I/O transaction forwarding for Root Port configuration.

Figure 20-132. PCI Secondary Status, I/O Limit, and I/O Base Registers (PECFGn_IOBASE)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31	DEPE	Detected Parity Error: 1 The PCI Express port received a Poisoned TLP on the Root Complex Device Secondary Bus	0b0	NA	R/W1C
30	SSE	Received System Error: 1 The PCI Express port received a non fatal/fatal Error Message on the Root Complex Device Secondary Bus	0b0	NA	R/W1C
29	RMA	Received Master Abort: 1 The PCI Express port received Completion with Unsupported Request on the Root Complex Device Secondary Bus	0b0	NA	R/W1C
28	RTA	Received Target Abort: 1 The PCI Express port received Completion Abort on the Root Complex Device Secondary Bus	0b0	NA	R/W1C
27	STA	Signaled Target Abort: 1 The PCI Express port signaled Completion Abort on the Root Complex Device Secondary Bus	0b0	NA	R/W1C
26:25		Reserved	0b00	NA	R
24	MDPE	Master Data Parity Error: 1 The PCI Express port received a Poisoned TLP on the Type1 Device Secondary Bus	0b0	NA	R/W1C
23:16		Reserved	0x00	NA	R
15:12	LRANGE	I/O Limit Address Range Supported	0x0	NA	R/W

11:8	LCAP	I/O Limit Addressing Capability: 32-bit access	0x1	NA	R
7:4	BRANGE	I/O Base Address Range Supported	0x0	NA	R/W
3:0	BCAP	I/O Base Addressing Capability: 32-bit access	0x1	NA	R

20.24.2.2 PCI Memory Limit and Memory Base Registers (PECFGn_MEMBAS)

These registers in a Root Port define a memory-mapped address range of 32 bits inside the 0-4GByte space, which is used to permit Endpoint-to-Endpoint communications without going upstream through the Root Port. This applicable for a PCI Express switch.

This memory zone in a Root Port only is the area between the two addresses defined by the BASE and LIMIT fields in this register.

If an Endpoint targets this address range, the Root Port ignores it and all associated transactions.

The address sent by the Endpoint to the Root Port must be outside the windows for upstream transactions.

If the BASE address is higher than the LIMIT address, transactions at any address are taken by the Root Port.

Figure 20-133. PCI Memory Limit and Memory Base Registers (PECFGn_MEMBAS)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	LIMIT	Memory Limit Address: 12 most significant bits of the Memory Limit Address defined as 32 bits **Note:** Memory Limit Address must be above the Memory Base Address.	0x000	N/A	R/W
19:16		Reserved	0x0	N/A	R
15:4	BASE	Memory Base Address: 12 most significant bits of the Memory Base Address defined as 32 bits	0x000	N/A	R/W
3:0		Reserved	0x0	N/A	R

20.24.2.3 PCI Prefetchable Memory Limit and Memory Base Registers (PECFGn_FTCHBAS)

These registers are used to control memory transaction forwarding for a Root Port configuration.

Functionally, the PCI Prefetchable Memory Limit High and Low registers can be viewed as a single 64-bit register. The PCI Prefetchable Memory Base High and Low registers can also be viewed as a single 64-bit register.

Figure 20-134. PCI Prefetchable Memory Limit and Memory Base Registers (PECFGn_FTCHBAS)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	PLRANG	Prefetchable Memory Limit	0x000	N/A	R/W
19:16	PLCAP	Prefetchable Memory Limit: 64-bit Addressing	0x1	N/A	R
15:4	PMBASE	Prefetchable Memory Base	0x000	N/A	R/W
3:0	PMCAP	Prefetchable Memory Base: 64-bit Addressing	0x1	N/A	R

20.24.2.4 PCI Prefetchable Memory Base High Register (PECFGn_FTCHBUP)

This register is used to control 64-bit memory transaction forwarding for a Root Port configuration.

Figure 20-135. PCI Prefetchable Memory Base High Register (PECFGn_FTCHBUP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	BASE	Prefetchable Memory Base Upper	0x0000_0000	N/A	R/W

20.24.2.5 PCI Prefetchable Memory Limit High Register (PECFGn_FTCHLUP)

This register is used to control 64-bit memory transaction forwarding for a Root Port configuration.

Figure 20-136. PCI Prefetchable Memory Limit High Register (PECFGn_FTCHLUP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LIMIT	Prefetchable Memory Limit Upper	0x0000_0000	N/A	R/W

20.24.2.6 PCI I/O Limit High and I/O Base High Registers (PECFGn_IOBLUP)

These registers are used to control I/O transaction forwarding for a Root Port configuration.

Figure 20-137. PCI I/O Limit High and I/O Base High Registers (PECFGn_IOBLUP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16	LIMIT	Upper I/O Limit	0x0000	N/A	R/W
15:0	BASE	Upper I/O Base	0x0000	N/A	R/W

20.24.2.7 PCI Root Port Expansion ROM Base Address Register (PECFGn_RTEROM)

This register is used by devices that need ROM code for port-specific initialization or system-boot functionality. The Expansion ROM BAR is very similar in operation to the standard base address register (BAR).

Both Type0 and Type1 header types have an Expansion ROM Base Address Register. However, this register is located at different offsets in the configuration space address map, depending on the PCI Header Type (Type0 = 0x030, Type1 = 0x038). The register definition and operation is the same for both Type0 and Type1 headers, but has been included in both sections of this document because of the different address offsets associated with the two header types.

Figure 20-138. PCI Root Port Expansion ROM Base Address Register (PECFGn_RTEROM)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:11	BAR	Root Port Expansion ROM Base Address: 32-bit BAR3	0x0_0000	N/A	R/W
10:1		Reserved	0b00_0000_0000	N/A	R
0	MSI	Expansion ROM Enable: 1 Enable (disabled at reset)	Software Initialization See PECFGn_RTEROMPA[ROME]	N/A	R

20.24.2.8 PCI Bridge Control Register (PECFGn_BCR)

This register provides extensions to PECFGn_CMDSTATUS. It provides many of the same controls for the secondary interface that are provided by PECFGn_CMDSTATUS for the primary interface.

Figure 20-139. PCI Bridge Control Register (PECFGn_BCR)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:23		Reserved	0b0_0000_0000	N/A	R
22	SBR	Secondary Bus Reset. Setting this bit triggers a Hot Reset on the corresponding PCI Express port	0b0	N/A	R/W
21:18		Reserved	0x0	N/A	R
17	SERRE	SERR Enable	0b0	N/A	R/W
16	PERE	Parity Error Response Enable	0b0	N/A	R/W
15:8	INTPN	Interrupt Pin: 0x00 - Not connected to INTx 0x01 - Connected to INTA 0x02 - Connected to INTB 0x03 - Connected to INTC 0x04 - Connected to INTD	Software Initialization See PECFGn_INTPPA[INTPN]	R	R
7:0	INTLN	Interrupt Line	0x00	R/W	R/W

20.25 PCI Configuration Space Capability Structures

The *PCI Express Specification* defines the following PCI configuration space capability structures:

- PM - PCI Express Power Management Capability Structure.
- MSI - PCI Express Message Signaled Interrupt Capability Structure.
- EC - PCI Express Capability Structure.
- VPD - PCI Express Vital Product Data Capability Structure.

All PCI capability structures are located in the PCI Express configuration space address map between address offsets 0x040 and 0x0FC.

All PCI capability structures are located and accessed by the system software using the address pointer link list mechanism defined as part of the PCI capability structure specifications. PECFGn_CAP[CAP] contains the 8-bit configuration space address pointer to the first PCI capability structure. Each capability structure contains a Next Pointer Register, which points to the next PCI capability structure in the linked list. The last PCI capability structure always has the value x'00' in its Next Pointer Register to signal the end of the PCI capability structure linked list.

The following sections describe the registers associated with the PCI capability structures supported by the PCI Express port (PM, MSI, EC, and VPD).

20.25.1 PCI Express Power Management (PM) Capability Structure

The PM capability structure registers are used to communicate and control the power management capabilities of all PCI Express ports.

Table 20-22. PCI Express Power Management (PM) Capability Structure Registers

Register	Address Doubleword Offset	PCI Express Attributes Endpoint only	PLB Attributes	Description	Page
PECFGn_PMCAP	0x040	R	R	PM Capability ID	603
		R	R	PM Next Pointer	603
		R	R	PM Capability	603
PECFGn_PMCSR	0x044	R/W	R/W	PM Status and Control Registers	604

20.25.1.1 PM Capability ID, Next Pointer, and Capability Registers (PECFGn_PMCAP)

The Power Management Capability Structure contains the:
- PCI-SIG defined Capability ID for the structure.
- Next Pointer Register, which points to the next MSI capability structure in the linked list.

This register is used to communicate the hardware-supported power management capabilities of the device to the operating system.

Figure 20-140. PM Capability ID, Next Pointer, and Capability Registers (PECFGn_PMCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:27	PME	PME Support: No generation or forwarding capability	0b0_0000	R	R
26	D2	D2 Support	0b0	R	R
25	D1	D1 Support	0b0	R	R
24:22	AUX	AUX Current: No PME Generation	0b000	R	R
21:18		Reserved	0b0000	R	R
17:16	VERS	Version	0b11	R	R
15:8	NPTR	Next Pointer	0x48	R	R
7:0	CAPID	Capability ID	0x01	R	R

20.25.1.2 PM Status and Control Register (PECFGn_PMCSR)

This register is used to communicate power management status and control information between the hardware and software.

Figure 20-141. PM Status and Control Register (PECFGn_PMCSR)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16		Reserved	0x0000	R	R
15	PMES	PME Status: No PME Generation	0b0	R	R
14:9		Reserved	00_0000	R	R
8	PMEE	PME Enable: No PME Generation	0b0	R	R
7:2		Reserved	00_0000	R	R
1:0	STATE	Power State	0b00	R/W	R/W

20.25.1.3 PCI Express Message Signaled Interrupt Capability Structure

The MSI Capability Structure registers are used to communicate and control the MSI capabilities of all PCI Express ports.

Table 20-23. Message Signaled Interrupt (MSI) Capability Structure Registers

Register	Address Doubleword Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_OMCAP	0x048	R	R	MSI Capability ID	604
		R	R	MSI Next Pointer	604
		R/W	R/W	MSI Message Control	604
PECFGn_OMMAL	0x04C	R/W	R/W	MSI Message Lower Address	605
PECFGn_OMMAU	0x050	R/W	R/W	MSI Message Upper Address	605
PECFGn_OMMDATA	0x054	R/W	R/W	MSI Message Data	606

20.25.1.4 MSI Message Control, Next Pointer, and Capability ID Register (PECFGn_OMCAP)

This register contains the:

- Message Control Register, which is used to communicate the port MSI requirements and control information between hardware and software
- Next Pointer Register, which points to the next (EC) capability structure in the linked list
- PCI-SIG defined Capability ID for the MSI capability structure

Figure 20-142. MSI Message Control, Next Pointer, and MSI Capability ID Register (PECFGn_OMCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24		Reserved	0x00	R	R
23	MSI64	64-Bit MSI Capability	0b1	R	R
22:20	MMEN	Multiple Message Enable: 000 1 message allocated 001 2 messages allocated 010 4 messages allocated All other values are reserved.	0b000	R/W	R/W
19:17	MMCAP	Multiple Message Capability: 4 messages	0b010	R	R
16	MSIE	MSI Enable	0b0	R/W	R/W
15:8	NPTR	Next Pointer	0x58	R	R
7:0	CAPID	Capability ID: MSI	0x05	R	R

20.25.1.5 MSI Message Lower Address Register (PECFGn_ OMMAL)

This register is used to communicate the software-assigned message address to the hardware that generates the messages.

Figure 20-143. MSI Message Lower Address Register (PECFGn_ OMMAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:2	MSIAL	MSI Message Lower Address	0x0000_0000	R/W	R/W
1:0		Reserved	0b00	R	R

20.25.1.6 MSI Message Upper Address Register (PECFGn_ OMMAU)

This register is used to communicate the software-assigned message address to the hardware that generates the messages (enable MSI 64-bit message address support).

Figure 20-144. MSI Message Upper Address Register (PECFGn_ OMMAU)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	MSIAU	MSI Message Upper Address	0x0000_0000	R/W	R/W

20.25.1.7 MSI Message Data Register (PECFGn_OMMDATA)

This register is used to communicate the software-specified message data to the hardware that generates the messages.

Figure 20-145. MSI Message Data Register (PECFGn_OMMDATA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16		Reserved	0x0000	R	R
15:0	DATA	Message Data	0x0000	R/W	R/W

20.25.1.8 PCI Express Capability Structure

This section describes the PCI Express Capability (EC) registers used for communicating and controlling the PCI Express capabilities.

Table 20-24. PCI Express Capability (EC) Structure Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_ECCAPID	0x058	R	R	EC Capability ID	606
		R	R	EC Next Pointer	606
		R	R	EC Capability	606
PECFGn_ECDEVCAP	0x05C	R	R	EC Device Capability	607
PECFGn_ECDEVCTL	0x060	R/W	R/W	EC Device Control	608
		R/W1C	R/W1C	EC Device Status	608
PECFGn_ECLNKCAP	0x064	R	R	EC Link Capability	609
PECFGn_ECLNKCTL	0x068	R/W	R/W	EC Link Control	610
		R	R	EC Link Status	610
PECFGn_ECSLTCAP	0x06C	NA	R	EC Slot Capability (Root Port only)	611
PECFGn_ECSLTCTL	0x070	NA	R/W	EC Status and Slot Control (Root Port only)	612
PECFGn_ECRTCTL	0x074	NA	R/W	EC Root Control (Root Port only)	613
PECFGn_CRSCS	0x076	NA	R/W	CRS Completion Status	613
PECFGn_ECRTSTA	0x078	NA	R/W1C	EC Root Status (Root Port only)	614

20.25.1.9 EC Capability, Capability ID, and Next Pointer Register (PECFGn_ECCAPID)

This register contains:

- PCI-SIG defined Capability ID for EC capability structure
- Capability information
- Next Pointer field, which terminates the capability structure linked list

This register describes the PCI Express port hardware type and basic capabilities to the software.

The firmware initializes this register by writing the corresponding fields of the associated programming access register (PECFG*n*_ECCAPIDPA). See *EC PCI Express Capability Programming Access Register (PECFGn_ECCAPIDPA)* on page 631 for details.

Figure 20-146. EC PCI Express Capability, Capability ID, and Next Pointer Reg (PECFGn_ECCAPID)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:30		Reserved	0b00	R	R
29:25	IRQN	Interrupt Message Number	Software Initialization See PECFGn_ECCAPIDPA[IRQN]	R	R
24	SLOT	Root Port: Slot Implemented Endpoint: Reserved	Software Initialization See PECFGn_ECCAPIDPA[SLOT]	N/A	R/W
23:20	TYPE	Device/Port Type 0000 PCI Express Endpoint 0001 Legacy PCI Express Endpoint 0100 Root Port All others reserved.	Copy of SDR0_PEn_DLPSET[PTYPE]	R	R
19:16	VER	Capability Version	0b0001	R	R
15:8	NPTR	Next Pointer	0x00	R	R
7:0	CAPID	PCI Express Capability ID	0x10	R	R

20.25.1.10 EC Device Capability Register (PECFGn_ECDEVCAP)

This register describes the PCI Express port-specific capabilities to the software. See *EC Device Capability Programming Access Register (PECFGn_ECDEVCAPPA)* on page 631 for more information.

Figure 20-147. EC Device Capability Register (PECFGn_ECDEVCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:28		Reserved	0x0	R	R
27:26	PWSCL	Captured Slot Power Limit Scale: Specify the scale used for the Captured Slot Power Limit Value: 00 1.0x 01 0.1x 10 0.01x 11 0.001x Bit valid for Endpoint configuration.	0b00	R	R
25:18	PWVAL	Captured Slot Power Limit Value: Specifies the upper limit on power supplied by the slot. The power limit (in watts) is calculated by multiplying the value in this field by the scale value specified in the PWSCL field. Bit valid for Endpoint configuration.	0x00	R	R
17:16		Reserved	0b00_0000	R	R

15	RBER	Role-Based Error Reporting: The PCI Express port implements the error reporting scheme of *PCI Express Base Specification* 1.0a and 1.1.	0b1	R	R
14:12		Reserved	0b000	R	R
11:9	L1LAT	Endpoint L1 Acceptable Latency (valid for Endpoint only)	Software Initialization See PECFGn_ECDEVCAPPA[L1LAT]	R	R
8:6	L0LAT	Endpoint L0s Acceptable Latency (valid for Endpoint only)	Software Initialization See PECFGn_ECDEVCAPPA[L0LAT]	R	R
5:3		Reserved	0b000	R	R
2:0	PSIZE	Upper Limit of Maximum Payload Size: 000 128-Byte Max payload 001 256-Byte Max payload 010 512-Byte Max payload All other values are reserved.	Software Initialization See PECFGn_ECDEVCAPPA[PSIZE]	R	R

20.25.1.11 EC Device Status and Control Status Registers (PECFGn_ECDEVCTL)

These registers provide PCI Express port-specific information to the software. They are used by the software to control the PCI Express port-specific parameters.

Figure 20-148. EC Device Status and Control Status Registers (PECFGn_ECDEVCTL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:22		Reserved	0b00_0000_0000	R	R
21	PTRA	Transactions Pending	0b0	R	R
20	AXPW	AUX Power Detected	0b0	R	R
19	UREQ	Unsupported Request Detected	0b0	R/W1C	R/W1C
18	FERR	Fatal Error Detected	0b0	R/W1C	R/W1C
17	NFER	Non Fatal Error Detected	0b0	R/W1C	R/W1C
16	CERR	Correctable Error Detected	0b0	R/W1C	R/W1C
15		Reserved	0b0	R	R
14:12	RSIZE	Maximum Effective Read Request Size: The upper limit is defined in SDR0_PEn_UTLSET1[MAXSZ]. 000 128 bytes 001 256 bytes 010 512 bytes 011 1024 bytes All other values are reserved.	0b000	R/W	R/W
11	NSNP	Enable No Snoop	0b1	R/W	R/W
10:8		Reserved	0b000	R	R

7:5	MPSZ	Maximum Effective Payload Size: The upper limit is defined in PECFG*n*_ECDEVCAP[PSIZE]. 000 128-Byte Max Payload 001 256-Byte Max Payload 010 512-Byte Max Payload All other values are reserved.	0b000	R/W	R/W
4	RXOE	Relaxed Ordering Enable	0b1	R/W	R/W
3	UREQE	Unsupported Request Reporting Enable	0b0	R/W	R/W
2	FERRE	Fatal Error Reporting Enable	0b0	R/W	R/W
1	NFERE	Non Fatal Error Reporting Enable	0b0	R/W	R/W
0	CERRE	Correctable Error Reporting Enable	0b0	R/W	R/W

20.25.1.12 EC Link Capability Register (PECFGn_ECLNKCAP)

This register provides PCI Express link-specific capabilities to the software. See *EC Link Capability Programming Access Register (PECFGn_ECLNKCAPPA)* on page 631 for more information.

Figure 20-149. EC Link Capability Register (PECFGn_ECLNKCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24	PNUM	Port Number	Software Initialization See PECFGn_ECLNKCAPPA[PNUM]	R	R
23:21		Reserved	0b00_0000	R	R
20	DLLARC	Data Link Layer Link Active Reporting Capable: For a Root Port, this bit is hardwired to 1 to indicate the reporting capability of the Data Link Control & Management State Machine. For an Endpoint, this bit is hardwired to 0.	(Root Port) 0b1 (Endpoint) 0b0	R	R
19	SUDERS	Surprise Down Error Reporting Capable: For a Root Port, this bit is hardwired to 1 to indicate the capability of detecting and reporting a Surprise Down Error condition. For an Endpoint, this bit is hardwired to 0.	(Root Port) 0b1 (Endpoint) 0b0	R	R
18		Reserved	0b0	R	R
17:15	L1LAT	L1 Exit Latency: From 1 to less than 2 microseconds	0b001	R	R
14:12	L0LAT	L0s Exit Latency: From 128 to less than 256 nanoseconds	0b010	R	R
11:10	PMSUP	Active State Link PM Support: L0 and L1 states supported	0b11	R	R

9:4	MLW	Maximum Link Width: 000001 - x1 width 000100 - x4 width 001000 - x8 width All other values are reserved	Copy of SDR0_PEn_DLPSET[LNKW]	R	R
3:0	MLS	Maximum Link Speed	0b0001	R	R

20.25.1.13 EC Link Status and Link Control Registers (PECFGn_ECLNKCTL)

The EC Link Status Register provides PCI Express link-specific information to the software.

The EC Link Control Register is used by the software to control the PCI Express link-specific parameters. See *EC Link Status and Control Programming Access Register (PECFGn_ECLNKCTLPA* on page 632 for more information/

Figure 20-150. EC Link Status and Link Control Registers (PECFGn_ECLNKCTL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:30		Reserved	0b000	R	R
29	DLLLA	Data Link Layer Link Active Status: Status of the Data Link Control and Management State Machine. It returns a 1 to indicate the DL_Active state, otherwise 0.	0b0	R	R
28	SCLK	Slot Clock Configuration: 0 Independent clock regardless of the presence of a reference on the connector. 1 Same physical reference clock that the platform provides on the connector.	Software Initialization See PECFGn_ECLNKCTLPA[SCLK]	R	R
27	TRN	Link Training	0b0	R	R
26		Reserved	0b0	R	R
25:20	WIDTH	Negotiated Link Width: 000001 - x1 width 000100 - x4 width 001000 - x8 width	0b00_0000	R	R
19:16	SPEED	Link Speed	0x1	R	R
15:8		Reserved	0x00	R	R
7	ESYNC	Extended Sync	0b0	R/W	R/W
6	CCLK	Common Clock Configuration	0b0	R/W	R/W
5	RTRN	Retrain Link	0b0	R/W	R/W
4	DISA	Link Disable	0b0	R/W	R/W
3	RCB128	Read Completion Boundary (RCB): Endpoint configuration: hardwired to 0. Root Complex configuration: hardwired to 1.	0b0 0b1	R NA	R R
2		Reserved	0b0	R	R

1:0	PMCTL	Active State Link PM Control: 00 Disabled 01 L0s entry enabled 10 L1 entry enabled 11 L0s and L1 enabled	Software Initialization See PECFGn_ECLNKCTLPA[PMCTL]	R/W	R/W

20.25.1.14 EC Slot Capability Register (PECFGn_ECSLTCAP)

This register describes the PCI Express slot-specific capabilities to the software. It is required for PCI Express Root Ports, optional for PCI Express Switch Ports, and is never present for PCI Express Endpoint devices.

The firmware initializes this register by writing the corresponding bit of PECFGn_ECSLTCAPPA. See *EC Slot Capability Programming Access Register (PECFGn_ECSLTCAPPA)* on page 632 for details.

Figure 20-151. EC Slot Capability Register (PECFGn_ECSLTCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:19	NUM	Physical Slot Number: This hardware-initialized field indicates the physical slot number attached to this port. This field must be assigned to a slot number that is globally unique within the chassis.	Software Initialization	NA	R
18	NOCCS	No Command Completed Support	Software Initialization	NA	R
17	EMIP	Electromechanical Interlock Present	Software Initialization	NA	R
16:15	PWSCL	Slot Power Limit Scale: 00 1.0x 01 0.1x 10 0.01x 11 0.001x	Software Initialization	NA	R
14:7	PWVAL	Slot Power Limit Value: Upper limit on power supplied by the slot. The power limit (in watts) is calculated by multiplying the value in this field by the value defined in the PWSCL field.	Software Initialization	NA	R
6	HPCAP	Hot-Plug Capable: 1 Slot capable of supporting Hot-Plug operations.	Software Initialization	NA	R
5	HPSUR	Hot-Plug Surprise: 1 A device present in this slot might be removed from the system without any prior notification.	Software Initialization	NA	R
4	PWIND	Power Indicator Present: 1 A Power Controller is implemented on the chassis for this slot.	Software Initialization	NA	R
3	ATIND	Attention Indicator Present: 1 An Attention Indicator is implemented on the chassis for this slot.	Software Initialization	NA	R
2	MRL	MRL Sensor Present: 1 An MRL Sensor is implemented on the chassis for this slot.	Software Initialization	NA	R
1	PWCTL	Power Controller Present: 1 A Power Controller is implemented for this slot.	Software Initialization	NA	R

0	ATBUT	Attention Button Present: 1 An Attention Button is implemented on the chassis for this slot.	Software Initialization	NA	R

20.25.1.15 EC Status and Slot Control Register (PECFGn_ECSLTCTL)

This register provides PCI Express slot-specific information to the software. It is used by the software to control the PCI Express slot-specific parameters.

It is only required for PCI Express Root Ports, and is never present for PCI Express Endpoints.

Figure 20-152. EC Slot Control Register (PECFGn_ECSLTCTL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:25		Reserved	0x_000	NA	R
24	DLLSC	Data Link Layer State Changed: 1 The value reported in PECFGn_ECLNKCTL[DLLLA] was changed. In response to a Data Link Layer State Changed event, software must read PECFGn_ECLNKCTL[DLLLA] to determine if the link is active before initiating configuration cycles to the Hot-Plug device.	0b0	NA	R/W1C
23	EMIS	Electromechanical Interlock Status: Hardwired to 0 for 460EX/EXr/GT.	0b0	NA	RO
22	PDST	Presence Detect State: 0 For 460EX/EXr/GT: Slot Empty	0b0	NA	R/W
21	MRLSS	MRL Sensor State: 0 For 460EX/EXr/GT: MRL Closed	0b0	NA	R/W
20	CMDC	Command Completed: Hardwired to 0 for 460EX/EXr/GT.	0b0	NA	
19	PDCH	Presence Detect Changed: 0 For 460EX/EXr/GT	0b0	NA	
18	MRLSC	MRL Sensor Changed: 0 For 460EX/EXr/GT	0b0	NA	
17	PWFD	Power Fault Detect: 0 For 460EX/EXr/GT	0b0	NA	
16	ATBP	Attention Button Pressed: 0 For 460EX/EXr/GT	0b0	NA	
15:13		Reserved	0b0000	NA	
12	DLLSCE	Data Link Layer State Changed Enable: 1 Enables software notification if PECFGn_ECLNKCTL[DLLLA] is changed.	0b0	NA	R/W
11	EMIC	Electromechanical Interlock Control: Unused	0b0	NA	R/W
10	PWCTL	Power Controller Control: Unused	0b0	NA	R/W
9:8	PWIND	Power Indicator Control: Unused	0b00	NA	R/W
7:6	ATIND	Attention Indicator Control: Unused	0b00	NA	R/W
5	HPIRQ	Hot-Plug Interrupt Enable: Unused	0b0	NA	R/W
4	SLIRQ	Command Completed Interrupt Enable: Unused	0b0	NA	R/W

3	DTCTE	Presence Detect Changed Enable: Unused	0b0	NA	R/W
2	MRLE	MRL Sensor Changed Enable: Unused	0b0	NA	R/W
1	PWFE	Power Fault Detected Enable: Unused	0b0	NA	R/W
0	ATBE	Attention Button Pressed Enable: Unused	0b0	NA	R/W

20.25.1.16 EC Root Control Register (PECFGn_ECRTCTL)

This register is used by the system software to control PCI Express Root Complex-specific parameters.

Figure 20-153. EC Root Control Register (PECFGn_ECRTCTL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
15:5		Reserved	0x000	NA	R
4	CRSV	CRS Software Visibility Enable: 1 Enables the Root Port to return Configuration Retry Status (CRS) Completion Status to software.	0b0	NA	R/W
3	PMEIE	PME Interrupt Enable	0b0	NA	R/W
2	FERRE	System Error on Fatal Error Enable	0b0	NA	R/W
1	NERRE	System Error on Non Fatal Error Enable	0b0	NA	R/W
0	CERRE	System Error on Correctable Error Enable	0b0	NA	R/W

20.25.1.17 CRS Completion Status Register (PECFGn_CRSCS)

This read-only register indicates that the Root Port is capable of returning CRS completion status to software.

Figure 20-154. CRS completion Status Register (PECFGn_CRSCS)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
15:1		Reserved	0x0000	NA	R
0	CRSV	CRS Software Visibility: 1 Indicates that the Root Port is capable of returning Configuration Request Retry Status (CRS) Completion Status to software.	0b1	NA	RO

20.25.1.18 EC Root Status Register (PECFGn_ECRTSTA)

This register provides PCI Express Root Complex-specific information to the software.

Figure 20-155. EC Root Status Register (PECFGn_ECRTSTA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:18		Reserved	0b0000_0000_0000_00	NA	R
17	PMEP	PME Pending: No PME Pending Queue	0b0	NA	R
16	PMES	PME Status	0b0	NA	R/W1C
15:0	PMEID	PME Requestor ID	0x0000	NA	R

20.25.1.19 PCI Express VPD Capability Structure

The VPD registers are used for communicating and controlling the Vital Product Data information that uniquely identifies hardware and, potentially, software elements of the system.

Table 20-25. PCI Express Vital Product Data (VPD) Capability Structure Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_VPDCAP	0x07C	R	R	VPD Capability ID	614
		R	R	VPD Next Pointer	614
		R	R	VPD Flag and Address	614
PECFGn_VPDDATA	0x080	R/W	R/W	VPD Data	615

20.25.1.20 VPD Flag, Address, Next Pointer, and Capability ID Register (PECFGn_VPDCAP)

This register contains:

- VPD flag bit and the address of the VPD
- Next Pointer field, which points to the Power Management Capability Structure in the linked list (see *PCI Express Power Management (PM) Capability Structure* on page 603 for more information)
- PCI SIG-defined Capability ID for the PCI Express VPD Capability Structure

Figure 20-156. VPD Flag, Address, Next Pointer, and Capability ID Register (PECFGn_VPDCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31	FLAG	VPD Flag: This bit indicates when the transfer of data between PECFGn_VPDDATA and the storage component is completed. This bit must be written at the same time as the ADDR field. See *PCI Express Port Reset Sequences* on page 516 for more information.	0b1	R/W	R/W
30:16	ADDR	VPD Address: The VPD address specifies the address of the VPD data to access. This field must be written at the same time as the FLAG bit.	0x0000	R/W	R/W
15:8	NPTR	Next Pointer	0x40	R	R
7:0	CAPID	Capability ID	0x03	R	R

20.25.1.21 VPD Data Register (PECFGn_VPDDATA)

This register contains information about the hardware, software, and microcode.

Figure 20-157. VPD Data Register (PECFGn_VPDDATA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0		Reserved (Root Port only).	0x0000_0000	R/W	R/W

20.26 PCI Express Extended Capabilities Structures

The *PCI Express Specification* extends the standard PCI 256-byte configuration space up to 4096 bytes. This PCI Express extended configuration space is used for the new PCI Express extended capability structures and features, and always starts at offset 0x100 in the PCI Express configuration space address map with a PCI Express Enhanced capability header. The PCI Express extended capability structures use the same link list technique used by the standard PCI capability structures (PCI Express extended capability IDs with Next Pointer Registers).

The PCI Express port implements two extended capability structures:

- AER - PCI Express Advanced Error Reporting Capability Structure.
- VSEC - PCI Vendor-Specific Extended Capability Structure.

All PCI capability structures are located in the PCI Express configuration space address map between address offsets 0x100 and 0x1FC. The following sections describe the configuration registers associated with the PCI Express extended capability structures.

20.26.1 PCI Express Advanced Error Reporting Extended Capability Structure

The PCI Express Advanced Error Reporting (AER) Extended Capability Structure is required for any PCI Express port that supports end-to-end CRC (ECRC). The following table describes the AER registers used for communicating and controlling this advanced error reporting capability.

Table 20-26. PCI Express Advanced Error Reporting (AER) Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_AERCAP	0x100	R	R	AER Next Capability Offset, Capability Version, and PCI Express EC ID	616
PECFGn_AERUESTA	0x104	R/W1CS	R/W1CS	AER Uncorrectable Error Status	617
PECFGn_AERUEMSK	0x108	R/WS	R/WS	AER Uncorrectable Error Mask	617
PECFGn_AERUESEV	0x10C	R/WS	R/WS	AER Uncorrectable Error Severity	618
PECFGn_AERCESTA	0x110	R/W1C	R/W1C	AER Correctable Error Status	619
PECFGn_AERCEMSK	0x114	R/WS	R/WS	AER Correctable Error Mask	619
PECFGn_AERCAPCTL	0x118	R/WS	R/WS	AER Capability and Control	620
PECFGn_AERHLOG1	0x11C	RS	RS	AER Header Log Register 1	620
PECFGn_AERHLOG2	0x120	RS	RS	AER Header Log Register 2	620
PECFGn_AERHLOG3	0x124	RS	RS	AER Header Log Register 3	620
PECFGn_AERHLOG4	0x128	RS	RS	AER Header Log Register 4	620
PECFGn_AERRTCTL	0x12C	NA	R/W	AER Root Error Control (only for Root Ports)	621
PECFGn_AERRTSTA	0x130	NA	R/W1CS	AER Root Error Status (only for Root Ports)	622
PECFGn_AERERRSID	0x134	NA	RS	AER Error Source Identification (only for Root Ports)	622

20.26.1.1 AER Next Capability Offset, Capability Ver, and EC ID Reg (PECFGn_AERCAP)

These registers contain:
- Next Capability Offset, which points to the VSEC extended capability structure in the linked list.
- Implemented Capability Version.
- PCI-SIG PCI Express-defined Extended Capability ID for the AER extended capability structure.

Figure 20-158. AER Next Capability Offset, Capability Version, and EC ID Register (PECFGn_AERCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	NPTR	Next Capability Offset	0x1F8	R	R
19:16	VER	Capability version	0x1	R	R
15:0	CAPID	PCI Express Extended Capability ID	0x0001	R	R

20.26.1.2 AER Uncorrectable Error Status Register (PECFGn_ AERUESTA)

This register contains error status for the individual uncorrectable error sources (non-fatal and fatal).

Figure 20-159. AER Uncorrectable Error Status Register (PECFGn_ AERUESTA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:21		Reserved	0b000_0000_0000	R	R
20	UR	Unsupported Request Error Status	0b0	R/W1CS	R/W1CS
19	ERR	ECRC Error Status	0b0	R/W1CS	R/W1CS
18	MTLP	Malformed TLP Status	0b0	R/W1CS	R/W1CS
17	RXOVL	Receiver Overflow Status	0b0	R/W1CS	R/W1CS
16	UNCMP	Unexpected Completion Status	0b0	R/W1CS	R/W1CS
15	CMPAB	Completer Abort Status	0b0	R/W1CS	R/W1CS
14	CMPTO	Completion Timeout Status	0b0	R/W1CS	R/W1CS
13	FCPE	Flow Control Protocol Error Status	0b0	R/W1CS	R/W1CS
12	PTLP	Poisoned TLP Status	0b0	R/W1CS	R/W1CS
11:5		Reserved	0b000_0000	R	R
4	DLPE	Data Link Protocol Error Status	0b0	R/W1CS	R/W1CS
3:0		Reserved	0x0	R	R

20.26.1.3 AER Uncorrectable Error Mask Register (PECFGn_AERUEMSK)

This register controls reporting of the individual uncorrectable error sources (non-fatal and fatal) of a PCI Express device to the PCI Express Root Port, using the PCI Express-defined error messages.

Figure 20-160. AER Uncorrectable Error Mask Register (PECFGn_AERUEMSK)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:21		Reserved	0b000_0000_0000	R	R
20	UR	Unsupported Request Error Mask	0b0	R/WS	R/WS
19	ERR	ECRC Error Mask	0b0	R/WS	R/WS
18	MTLP	Malformed TLP Mask	0b0	R/WS	R/WS
17	RXOVL	Receiver Overflow Mask	0b0	R/WS	R/WS
16	UNCMP	Unexpected Completion Mask	0b0	R/WS	R/WS
15	CMPAB	Completer Abort Mask	0b0	R/WS	R/WS
14	CMPTO	Completion Timeout Mask	0b0	R/WS	R/WS
13	FCPE	Flow Control Protocol Error Mask	0b0	R/WS	R/WS
12	PTLP	Poisoned TLP Mask	0b0	R/WS	R/WS
11:5		Reserved	0b000_0000	R	R
4	DLPE	Data Link Protocol Error Mask	0b0	R/WS	R/WS
3:0		Reserved	0x0	R	R

20.26.1.4 AER Uncorrectable Error Severity Register (PECFGn_AERUESEV)

This register controls whether individual uncorrectable errors are reported as fatal or non fatal. Individual errors are reported as non fatal if the ERR bit is set to 0, and as fatal if the bit is set to 1.

Figure 20-161. AER Uncorrectable Error Severity Register (PECFGn_AERUESEV)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:21		Reserved	0b000_0000_0000	R	R
20	UR	Unsupported Request Error Severity	0b0	R/WS	R/WS
19	ERR	ECRC Error Severity 0 Non fatal 1 Fatal	0b0	R/WS	R/WS
18	MTLP	Malformed TLP Severity	0b1	R/WS	R/WS
17	RXOVL	Receiver Overflow Severity	0b1	R/WS	R/WS
16	UNCMP	Unexpected Completion Severity	0b0	R/WS	R/WS
15	CMPAB	Completer Abort Severity	0b0	R/WS	R/WS
14	CMPTO	Completion Timeout Severity	0b0	R/WS	R/WS
13	FCPE	Flow Control Protocol Error Severity	0b1	R/WS	R/WS
12	PTLP	Poisoned TLP Severity	0b0	R/WS	R/WS
11:5		Reserved	0b000_0000	R	R
4	DLPE	Data Link Protocol Error Severity	0b1	R/WS	R/WS
3:0		Reserved	0x0	R	R

20.26.1.5 AER Correctable Error Status Register (PECFGn_ AERCESTA)

This register contains the error status for the individual correctable error sources.

Figure 20-162. AER Correctable Error Status Register (PECFGn_ AERCESTA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:14		Reserved	0x0_0000	R	R
13	ANFES	Advisory Non Fatal Error Status:	0b1	R/W1CS	R/W1CS
12	RTO	Replay Timer Timeout Status	0b0	R/W1CS	R/WC1S
11:9		Reserved	0b000	R	R
8	ROVR	Replay NUM Rollover Status	0b0	R/W1CS	R/WC1S
7	BDLP	Bad DLLP Status	0b0	R/W1CS	R/WC1S
6	BTLP	Bad TLP Status	0b0	R/W1CS	R/WC1S
5:1		Reserved	0b0_0000	R	R
0	RXER	Receiver Error Status	0b0	R/W1CS	R/WC1S

20.26.1.6 AER Correctable Error Mask Register (PECFGn_AERCEMSK)

This register controls reporting of the individual correctable error sources of a PCI Express device to the PCI Express Root Port, using the PCI Express-defined error messages.

Figure 20-163. AER Correctable Error Mask Register (PECFGn_AERCEMSK)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:14		Reserved	0x0_0000	R	R
13	ANFEM	Advisory Non Fatal Error Mask: This bit is set by default to enable compatibility with software that does not recognize Role-Based Error Reporting	0b1	R/WS	R/WS
12	RTO	Replay Timer Timeout Mask	0b0	R/WS	R/WS
11:9		Reserved	0b000	R	R
8	ROVR	Replay NUM Rollover Mask	0b0	R/WS	R/WS
7	BDLP	Bad DLLP Mask	0b0	R/WS	R/WS
6	BTLP	Bad TLP Mask	0b0	R/WS	R/WS
5:1		Reserved	0b0_0000	R	R
0	RXER	Receiver Error Mask	0b0	R/WS	R/WS

20.26.1.7 AER Capability and Control Register (PECFGn_AERCAPCTL)

This register contains the First Error Pointer (FERP) field and also controls and reports the state of the ECRC generation capabilities.

The FERP field identifies the bit position of the first error reported in the AER Uncorrectable Error Status Register (PECFGn_AERUESTA). See *AER Uncorrectable Error Status Register (PECFGn_ AERUESTA)* on page 617 for details. If multiple uncorrectable errors occur on the same clock cycle, the PCI Express port uses the following error priority for setting the FERP field value (listed from lowest to highest priority):

DLLPE
FCPE
CMPL_TIMEOUT
REC_OVERFLOW
UR
SIG_CMPL_ABORT
UNEXPECTED_CMPL
REC_POISONED_TLP
MALFORMED_TLP
ECRC_ERROR

Figure 20-164. AER Capability and Control Register (PECFGn_AERCAPCTL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:9		Reserved	0x00_0000	R	R
8	CHKE	ECRC Check Enable	0b0	R/WS	R/WS
7	CHKA	ECRC Check Capable	Software Initialization See PECFGn_AERCAPCTLPA[CHKA]	R	R
6	GEN	ECRC Generation Enable	0b0	R/WS	R/WS
5	GENC	ECRC Generation Capable	Software Initialization See PECFGn_AERCAPCTLPA[GENC]	R	R
0:4	FERP	First Error Pointer	0b00000	RS	RS

20.26.1.8 AER Header Log Registers (PECFGn_AERHLOG1:4)

These registers capture the TLP header of the packet that corresponds to the first loggable non masked uncorrectable error.

The TLP header is captured in the four AER header log registers such that:

- Header Log Register 1 (PECFGn_AERHLOG1[HDLGR1]) contains TLP header bytes 0–3
  - Where TLP header byte 3 is the least significant byte of the AER header log register
- Header Log Register 2 (PECFGn_AERHLOG2[HDLGR2]) contains TLP header bytes 4–7
  - Where TLP header byte 7 is the least significant byte of the AER header log register
- Header Log Register 3 (PECFGn_AERHLOG3[HDLGR3]) contains TLP header bytes 8–11
  - Where TLP header byte 11 is the least significant byte of the AER header log register
- Header Log Register 4 (PECFGn_AERHLOG4[HDLGR4]) contains TLP header bytes 12–15
  - Where TLP header byte 15 is the least significant byte of the AER header log register
  - This value in this register is undefined if 12-byte TLP headers are logged.

Figure 20-165. AER Header Log Registers (PECFGn_AERHLOG1:4)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	HDLGR1	Header Log Register 1: TLP header bytes 0:3	0x0000_0000	RS	RS
31:0	HDLGR2	Header Log Register 2: TLP header bytes 4:7	0x0000_0000	RS	RS
31:0	HDLGR3	Header Log Register 3: TLP header bytes 8:11	0x0000_0000	RS	RS
31:0	HDLGR4	Header Log Register 4: TLP Header Bytes 12:15 Undefined for 12-byte headers	0x0000_0000	RS	RS

20.26.1.9 AER Root Error Control Register (PECFGn_AERRTCTL)

This register provides additional control of the Root Port response to correctable, non fatal, and fatal error messages. It enables and disables AER interrupt generation for each error type.

Figure 20-166. AER Root Error Control Register (PECFGn_AERRTCTL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:3		Reserved	0x0000_000	NA	R
2	FERRE	Fatal Error Reporting Enable	0b0	NA	R/W
1	NFERE	Non Fatal Error Reporting Enable	0b0	NA	R/W
0	CERRE	Correctable Error Reporting Enable	0b0	NA	R/W

20.26.1.10 AER Root Error Status Register (PECFGn_AERRTSTA)

This register contains the status of all errors (correctable, non fatal, and fatal) in the PCI Express hierarchy of the Root Port device (both internal Root Port errors and upstream error messages). It also contains the AER Message Signaled Interrupt (MSI) number associated with the AER reporting interrupt.

Figure 20-167. AER Root Error Status Register (PECFGn_AERRTSTA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:27	MSIN	Advanced Error Message Signaled Interrupt Number	Software Initialization See PECFGn_AERRTSTAPA[MSIN]	NA	R
26:7		Reserved	0x0_0000	NA	R
6	RFERR	Fatal Error Message Received	0b0	NA	R/W1CS
5	RNFER	Non Fatal Error Message Received	0b0	NA	R/W1CS
4	FIRST	First Uncorrectable Fatal	0b0	NA	R/W1CS
3	MULTI	Multiple Fatal and/or Non Fatal Errors Received	0b0	NA	R/W1CS
2	FNFER	Fatal and/or Non Fatal Errors Received	0b0	NA	R/W1CS
1	RMERC	Multiple Correctable Errors Received	0b0	NA	R/W1CS
0	RERRC	Correctable Error Received	0b0	NA	R/W1CS

20.26.1.11 AER Error Source Identification Register (PECFGn_AERERRSID)

This register identifies the source (requestor ID) of the first correctable and uncorrectable (non fatal and fatal) errors reported in the AER Root Error Status Register (PECFGn_AERRTSTA). Two sets of source ID signals feed this register, based on the origin of the error message:

- Source ID for internal Root Port-generated errors
- Source ID from upstream error messages

The PCI Express port always gives upstream message errors priority over internal Root Port errors if both errors occur on the same cycle (the assumption is that the upstream message-based errors must have occurred earlier in time).

Figure 20-168. AER Error Source Identification Register (PECFGn_AERERRSID)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16	FNF	Fatal and/or Non Fatal Error Source Identification	0x0000	NA	RS
15:0	CORR	Correctable Error Source Identification	0x0000	NA	RS

20.26.1.12 Vendor-Specific Extended Capability (VSEC) Structure

The Vendor-Specific Extended Capability (VSEC) Structure allows defining vendor-specific registers. The following table describes the registers used for communicating and controlling this structure.

Table 20-27. PCI Express Vendor-Specific Capability (VSEC) Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_VSECCAP	0x1F8	R	R	VSEC Next Capability Offset, Capability Version, and EC ID	623
PECFGn_VSECID	0x1FC	R	R	VSEC ID	623
		R	R	VSEC Revision	623
		R	R	VSEC Length	623

20.26.1.13 VSEC Next Capability Offset, Capability Ver, and EC ID Reg (PECFGn_VSECCAP)

This register contains:
- Next Capability Offset, which terminates the linked list of the extended capability structures
- Implemented Capability Version
- PCI-SIG-defined Extended Capability ID for the VSEC capability structure

Figure 20-169. VSEC Next Capability Offset, Capability Version, and EC ID Reg (PECFGn_VSECCAP)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	NPTR	Next Capability Offset	0x000	R	R
19:16	VER	VSEC Capability Version	0x1	R	R
15:0	CAPID	VSEC Extended Capability ID	0x000B	R	R

20.26.1.14 VSEC Length, Revision, and ID Register (PECFGn_VSECID)

This register contains the following fields:

- The LEN field indicates that the entire VSEC structure (including the PCI Express Enhanced Capability Header, the Vendor-Specific Header and the Vendor-Specific Registers) requires 576 bytes (From address offset 0x1F8 to 0x434).
- The REV field indicates the version of the VSEC Structure. Software must qualify both the Vendor ID (in (PECFGn_VENDEVID[VENDEVID]) - on page 592 - and the VSEC ID in this register before interpreting the REV field.
- The VSECID field indicates the type and format of the VSEC structure. Software must qualify the Vendor ID (PECFGn_VENDEVID[VENDEVID]) - on page 592 - before interpreting the VSECID field.

Figure 20-170. VSEC Length, Revision, and ID Register (PECFGn_VSECID)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	LEN	VSEC Length	0x240	R	R
19:16	REV	VSEC Revision	Software Initialization See PECFGn_VSECIDPA[REV]	R	R
15:0	VSECID	VSEC ID	Software Initialization See PECFGn_VSECIDPA[ID]	R	R

20.26.2 Programming Access Registers

The following registers, located from address offset 0x200 to 0x338 within the configuration space, are read/write registers. They enable the software to program specific configuration registers located below address offset 0x200, which are defined as read only registers by the PCI Express standards.

Table 20-28. PCI Express Programming Access Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_VENDEVIDPA	0x200	R/W	R/W	PCI Device ID and Vendor ID Programming Access	625
PECFGn_REVIDCLASSPA	0x208	R/W	R/W	PCI Class Codes, Revision ID Programming Access	625
PECFGn_BAR0LMPA	0x210	R/W	R/W	PCI Base Address Register 0 Low Mask Programming Access	626
PECFGn_BAR0HMPA	0x214	R/W	R/W	PCI Base Address Register 0 High Mask Programming Access	626
PECFGn_BAR1MPA	0x218	R/W	R/W	PCI Base Address Register 1 Mask Programming Access	627
PECFGn_BAR2LMPA	0x220	R/W	R/W	PCI Base Address Register 2 Low Mask Programming Access	627
PECFGn_BAR2HMPA	0x224	R/W	R/W	PCI Base Address Register 2 High Mask Programming Access	628
PECFGn_SBSYSVIDPA	0x22C	R/W	R/W	PCI Subsystem ID/Vendor ID Programming Access	628
PECFGn_EROMBAPA	0x230	R/W	R/W	PCI Endpoint Expansion ROM Base Address Mask (BAR3) Programming Access	629
PECFGn_CAPPA	0x234	R/W	R/W	PCI Capability Pointer Programming Access	629
PECFGn_RTEROMPA	0x238	R/W	R/W	PCI Root Port Expansion ROM Base Address Mask (BAR3) Programming Access	630
PECFGn_INTPPA	0x23C	R/W	R/W	PCI Interrupt Pin Programming Access	630
PECFGn_ECCAPIDPA	0x258	R/W	R/W	EC PCI Express Capability Programming Access	631
PECFGn_ECDEVCAPPA	0x25C	R/W	R/W	EC Device Capability Programming Access	631
PECFGn_ECLNKCAPPA	0x264	R/W	R/W	EC Link Capability Programming Access	631
PECFGn_ECLNKCTLPA	0x268	R/W	R/W	EC Link Status and Link Control Programming Access	632
PECFGn_ECSLTCAPPA	0x26C	R/W	R/W	EC Slot Capability Programming Access	632
PECFGn_AERCAPCTLPA	0x318	R/W	R/W	AER Capability and Control Programming Access	633
PECFGn_AERRTSTAPA	0x330	R/W	R/W	AER Root Error Status Programming Access	633
PECFGn_VSECIDPA	0x338	R/W	R/W	VSEC Revision and VSEC ID Programming Access	633

20.26.2.1 PCI Device ID and Vendor ID Programming Access Register (PECFGn_VENDEVIDPA)

Figure 20-171. Vendor Length, Revision, ID Register (PECFGn_VENDEVIDPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16	DID	Device ID: Programming access for PECFGn_VENDEVID[DEVID]	0x0000	R/W	R/W
15:0	VID	Vendor ID: Programming access for PECFGn_VENDEVID[VENDEVID]	0x0000	R/W	R/W

20.26.2.2 PCI Class Codes and Revision ID Prog Access Registers (PECFGn_REVIDCLASSPA)

Figure 20-172. PCI Class Codes and Revision Prog ID Access Register (PECFGn_REVIDCLASSPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24	BASE	Base Class Code: Programming access for PECFGn_REVIDCLASS[BASE]	0x00	R/W	R/W
23:16	SUB	Sub-Class Code: Programming access for PECFGn_REVIDCLASS[SUB]	0x00	R/W	R/W
15:8	INT	Interface Class Code: Programming access for PECFGn_REVIDCLASS[INT]	0x00	R/W	R/W
7:0	RID	Revision ID: Programming access for PECFGn_REVIDCLASS[RID]	0x00	R/W	R/W

20.26.2.3 PCI Base Address Register 0 Low Mask Programming Access (PECFGn_BAR0LMPA)

Figure 20-173. PCI Base Address Register 0 Low Mask Programming Access (PECFGn_BAR0LMPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	BAL	Base Address 0 Region (64-bit BAR0) Low Mask: Programming access for PECFGn_BAR0L[BAL] Bits 1:43 of the 64-bit BAR0 Region can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 1MB (2^{20}) to 2^{63}: 0000000000.....0000000000 - 2^{63}B 1000000000.....0000000000 - 2^{62}B 1100000000.....0000000000 - 2^{61}B --- 1111111111.....1111111100 - 2^{22}B 1111111111.....1111111110 - 2^{21}B 1111111111.....1111111111 - 2^{20}B	0x000	R/W	R/W
19:4		Reserved	0x0000	R	R
3	PEF	Prefetchable Bit: Programming access for PECFGn_BAR0L[PEF]	0b0	R/W	R/W
2:1	LT	Type Field: Programming access for PECFGn_BAR0L[LT]: 00 32-bit address memory space (not supported) 10 64-bit address memory space All other values are reserved.	0b10	R/W	R/W
0		Reserved	0b0	R	R

20.26.2.4 PCI Base Address Register 0 High Mask Programming Access (PECFGn_BAR0HMPA)

Figure 20-174. PCI Base Address Register 0 High Mask Programming Access (PECFGn_BAR0HMPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31		Reserved	0b0	R	R
30:0	BAH	Base Address 0 High (64-bit BAR0) Mask: Programming access for PECFGn_BAR0H: BAR0 Region Mask bits (1:31). See explanation for PECFGn_BAR0LMPA[BAL].	0x0000_0000	R/W	R/W

20.26.2.5 PCI Base Address Register 1 Mask Programming Access (PECFGn_BAR1MPA)

Figure 20-175. PCI Base Address Register 1 Mask Programming Access (PECFGn_BAR1MPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16		Reserved	0x0000	R	R
15:8	BAR1	Base Address 1 (32-bit BAR1) Mask: Programming access for PECFGn_BAR1[BAR1] for Endpoint, or PECFGn_RTBAR1[BAR1] for Root Port. Bits 15:8 of the BAR1 Region can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 128B (2^7) to 32KB (2^{15}): 00000000 - 2^{15}B 10000000 - 2^{14}B 11000000 - 2^{13}B --- 11111100 - 2^9B 11111110 - 2^8B 11111111 - 2^7B	0x00	R/W	R/W
7:1		Reserved	0x00	R	R
0	MMIOS	Space Indicator: Programming access for PECFGn_BAR1[IO]: 0 Memory 1 I/O	0b1	R/W	R/W

20.26.2.6 PCI Base Address Register 2 Low Mask Programming Access (PECFGn_BAR2LMPA)

Figure 20-176. PCI Base Address Register 2 Low Mask Programming Access (PECFGn_BAR2LMPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	BAL	Base Address 2 Low (64-bit BAR2) Mask: Programming access for PECFGn_BAR2L[BAL] for Endpoint, or PECFGn_RTBAR2L[BAR2] for Root Port: Bits 1:43 of the 64-bit BAR2 Region can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 1MB (2^{20}) to 2^{63}: 0000000000.....0000000000 - 2^{63}B 1000000000.....0000000000 - 2^{62}B 1100000000.....0000000000 - 2^{61}B --- 1111111111.....1111111100 - 2^{22}B 1111111111.....1111111110 - 2^{21}B 1111111111.....1111111111 - 2^{20}B	0x000	R/W	R/W
19:4		Reserved	0x0000	R	R
3	PEF	Prefetchable Bit: Programming access for PECFGn_BAR2L[PEF]	0b0	R/W	R/W

2:1	BAT	Base Address Type: Programming access for PECFGn_BAR2L[LT]: 00 32-bit address memory space (not supported) 10 64-bit address memory space Other values are reserved.	0b10	R/W	R/W
0		Reserved	0b0	R	R

20.26.2.7 PCI Base Address Register 2 High Mask Programming Access (PECFGn_BAR2HMPA)

Figure 20-177. PCI Base Address Register 2 High Mask Programming Access (PECFGn_BAR2HMPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31		Reserved	0	R	R
30:0	BAH	Base Address 2 High (64-bit BAR2) Mask: Programming access for PECFGn_BAR2H[BAH] for Endpoint, or PECFGn_RTBAR2H[BAH] for Root Port: BAR2 Region Mask bits (1:31). See explanation for PECFGn_BAR2LMPA[BAL].	0x0000_0000	R/W	R/W

20.26.2.8 PCI Subsystem ID/Vendor ID Programming Access Register (PECFGn_SBSYSVIDPA)

Figure 20-178. PCI Subsystem ID/Vendor ID Programming Access Register (PECFGn_SBSYSVIDPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16	SID	Subsystem ID: Programming access for PECFGn_SBSYSVID[SID].	0x0000	R/W	R/W
15:0	SVID	Subsystem Vendor ID: Programming access for PECFGn_SBSYSVID[SVID].	0x0000	R/W	R/W

20.26.2.9 PCI Endpoint Expansion ROM Base Addr Mask Prog Access (PECFGn_EROMBAPA)

This applies to an Endpoint only.

Endpoint is defined by SDR0_PEn_DLPSET[PTYPE].

Figure 20-179. PCI Endpoint Expansion ROM Base Addr Mask Prog Access (PECFGn_EROMBAPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24		Reserved	0x00	R	R
23:11	BAR	Endpoint Expansion ROM Base Address (32-bit BAR3) Mask: Programming access for PECFGn_EROMBA[BAR]: Bits 23:11 of the Endpoint Expansion ROM Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 2KB (2^{11}) to 16MB (2^{24}): 0000000000000 - 2^{24}B 1000000000000 - 2^{23}B 1100000000000 - 2^{22}B --- 1111111111100 - 2^{13}B 1111111111110 - 2^{12}B 1111111111111 - 2^{11}B	0x0000	R/W	R/W
10:1		Reserved	0x000	R	R
0	ROME	ROM Enabled: Programming access if Endpoint: 0 ROM Expansion space mechanism disabled 1 ROM Expansion space mechanism enabled	See SDR0_PEn_DLPSET [PTYPE]	N/A	R/W

20.26.2.10 PCI Capability Pointer Programming Access Register (PECFGn_CAPPA)

Figure 20-180. PCI Capability Pointer Programming Access Register (PECFGn_CAPPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:8		Reserved	0x000000	R	R
7:0	CAP	Capability Pointer: Programming access for PECFGn_CAP[CAP] Supported values are: 0x40 - Power Management Capabilities 0x7C - Vital Product Data Capabilities	0x7C	R/W	R/W

20.26.2.11 PCI Root Port Expansion ROM Base Addr Mask Prog Access (PECFGn_RTEROMPA)

This applies to a Root Port only.

Root Port is defined by SDR0_PEn_DLPSET[PTYPE].

Figure 20-181. PCI Root Port Only Expansion ROM Base Addr Mask Prog Access (PECFGn_RTEROMPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24		Reserved	0x00	R	R
23:11	BAR	Root Port Expansion ROM Base Address (32–bit BAR3) Mask: Programming access for PECFGn_RTEROM[BAR]: Bits 23:11 of the Root Port Expansion ROM Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 2KB (2^{11}) to 16MB (2^{24}): 0000000000000 - 2^{24}B 1000000000000 - 2^{23}B 1100000000000 - 2^{22}B --- 1111111111100 - 2^{13}B 1111111111110 - 2^{12}B 1111111111111 - 2^{11}B	0x0000	N/A	R/W
10:1		Reserved	0x000	R	R
0	ROME	Expansion ROM Enabled: Programming access if Root Port: 0 ROM Expansion space mechanism disabled 1 ROM Expansion space mechanism enabled	See SDR0_PEn_DLPSET[PTYPE]	N/A	R/W

20.26.2.12 PCI Interrupt Pin Programming Access Register (PECFGn_INTPPA)

Figure 20-182. PCI Interrupt Pin Programming Access Register (PECFGn_INTPPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:16		Reserved	0x0000	R	R
15:8	INTPN	Interrupt Pin: Programming access for PECFGn_INTLNPN[INTPN]. Supported Encoding Values Are: 0x00 - No INTx interrupt. 0x01 - INTA interrupt. 0x02 - INTB interrupt. 0x03 - INTC interrupt. 0x04 - INTD interrupt. All other values are reserved.	0x00	R/W	R/W

7:0		Reserved	0x00	R	R

20.26.2.13 EC PCI Express Capability Programming Access Register (PECFGn_ECCAPIDPA)

Figure 20-183. EC PCI Express Capability Programming Access Register (PECFGn_ECCAPIDPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:30		Reserved	0b00	R	R
29:25	IRQN	Interrupt Message Number Programming: Update PECFGn_ECCAPID[IRQN]	0b0_0000	R/W	R/W
24	SLOT	Slot Implemented Programming Access: Update PECFGn_ECCAPID[SLOT]	0b0	R/W	R/W
23:0		Reserved	0x00_0000	R	R

20.26.2.14 EC Device Capability Programming Access Register (PECFGn_ECDEVCAPPA)

Figure 20-184. EC Device Capability Programming Access Register (PECFGn_ECDEVCAPPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:12		Reserved	0x0_0000	R	R
11:9	L1LAT	Endpoint L1 Acceptable Latency: Programming access for PECFGn_ECDEVCAP[L1LAT]	0b000	R/W	R/W
8:6	L0LAT	Endpoint L0s Acceptable Latency: Programming access for PECFGn_ECDEVCAP[L0LAT]	0b000	R/W	R/W
5:3		Reserved	0b000	R	R
2:0	PSIZE	Maximum Payload Size: Programming access for PECFGn_ECDEVCAP[PSIZE] 000 128-Byte Max payload 001 256-Byte Max payload 010 512-Byte Max payload All other values are reserved.	0b000	R/W	R/W

EC Link Capability Programming Access Register (PECFGn_ECLNKCAPPA)

Figure 20-185. EC Link Capability Programming Access Register (PECFGn_ECLNKCAPPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24	PNUM	Port Number: Programming access for PECFGn_ECLNKCAP[PNUM]	0x00	R/W	R/W
23:0		Reserved	0x00_0000	R	R

20.26.2.15 EC Link Status and Control Programming Access Register (PECFGn_ECLNKCTLPA

Figure 20-186. EC Link Status and Control Programming Access Register (PECFGn_ECLNKCTLPA

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:29		Reserved	0b000	R	R
28	SCLK	Slot Clock Configuration: Programming access for PECFGn_ECLNKCTL[SCLK]	0b0	R/W	R/W
27:2		Reserved	0x000_0000	R	R
1:0	PMCTL	Active State Link PM Control: Programming access for PECFGn_ECLNKCTL[PMCTL]: 00 Disabled 01 L0s entry enabled	0b00	R/W	R/W

20.26.2.16 EC Slot Capability Programming Access Register (PECFGn_ECSLTCAPPA)

Figure 20-187. EC Slot Capability Programming Access Register (PECFGn_ECSLTCAPPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:19	NUM	Physical Slot Number: Programming access for PECFGn_ECSLTCAP[NUM]	0x0000	NA	R/W
18	NCCS	No Command Completed Support: Programming access for PECFGn_ECSLTCAP[NOCCS]	0b0	NA	R/W
17	EMIP	Electromechanical Interlock Present: Programming access for PECFGn_ECSLTCAP[EMIP]	0b0	NA	R/W
16:15	PWSCL	Slot Power Limit Scale: Programming access for PECFGn_ECSLTCAP[PWSCL]	0b00	NA	R/W
14:7	PWVAL	Slot Power Limit Value: Programming access for PECFGn_ECSLTCAP[PWVAL]	0x00	NA	R/W
6	HPCAP	Hot-Plug Capable: Programming access for PECFGn_ECSLTCAP[HPCAP]	0b0	NA	R/W
5	HPSUR	Hot-Plug Surprise: Programming access for PECFGn_ECSLTCAP[HPSUR]	0b0	NA	R/W
4	PWIND	Power Indicator Present: Programming access for PECFGn_ECSLTCAP[PWIND]	0b0	NA	R/W
3	ATIND	Attention Indicator Present: Programming access for PECFGn_ECSLTCAP[ATIND]	0b0	NA	R/W
2	MRL	MRL Sensor Present: Programming access for PECFGn_ECSLTCAP[MRL]	0b0	NA	R/W
1	PWCTL	Power Controller Present: Programming access for PECFGn_ECSLTCAP[PWCTL]	0b0	NA	R/W
0	ATBUT	Attention Button Present: Programming access for PECFGn_ECSLTCAP[ATBUT]	0b0	NA	R/W

20.26.2.17 AER Capability and Ctrl Programming Access Register (PECFGn_AERCAPCTLPA)

Figure 20-188. AER Capability and Ctrl Programming Access Register (PECFGn_AERCAPCTLPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:8		Reserved	0x00_0000	R	R
7	CHKA	ECRC Check Capable: Programming access for PECFGn_AERCAPCTL[CHKA]	0b0	R/W	R/W
6		Reserved	0b0	R	R
5	GENC	ECRC Generation Capable: Programming access for PECFGn_AERCAPCTL[GENC]	0b0	R/W	R/W
0:4	FERP	First Error Pointer	0b00000	RS	RS

20.26.2.18 AER Root Error Status Programming Access Register (PECFGn_AERRTSTAPA)

Figure 20-189. AER Root Error Status Programming Access Register (PECFGn_AERRTSTAPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:27	MSIN	Advanced Error Interrupt Message Number: Programming access for PECFGn_AERRTSTA[MSIN]	0b0_0000	NA	R/W
26:0		Reserved	0x000_0000	NA	R

20.26.2.19 VSEC Revision and VSEC ID Programming Access Register (PECFGn_VSECIDPA)

Figure 20-190. VSEC Revision and VSEC ID Programming Access Register (PECFGn_VSECIDPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20		Reserved	0x000	R	R
19:16	REV	VSEC Revision: Programming access for PECFGn_VSECID[REV]	0x0	R/W	R/W
15:0	ID	VSEC ID: Programming access for PECFGn_VSECID[VSECID]	0x0000	R/W	R/W

20.26.3 VSEC PIM and POM Registers

These registers are used to define address translation between the PCI Express port and PLB bus space.

Table 20-29. PCI Express VSEC Address Translation Registers

Register	Address Double-word Offset	PCI Express Attributes Endpoint Only	PLB Attributes	Description	Page
PECFGn_PIMEN	0x33C	R/W	R/W	InRegion Valid Programming Access	635
PECFGn_PIM0LAL	0x340	R/W	R/W	PIM0 Address Low for BAR0	635
PECFGn_PIM0LAH	0x344	R/W	R/W	PIM0 Address High for BAR0	636
PECFGn_PIM1LAL	0x348	R/W	R/W	PIM1 Address Low for BAR0	636
PECFGn_PIM1LAH	0x34C	R/W	R/W	PIM1 Address High for BAR0	636
PECFGn_PIM01SAL	0x350	R/W	R/W	PIM01 Size Address Low for BAR0	637
PECFGn_PIM01SAH	0x354	R/W	R/W	PIM01 Size Address High for BAR0	637
PECFGn_PIM2LAL	0x358	R/W	R/W	PIM2 Address Low for BAR1	638
PECFGn_PIM2LAH	0x35C	R/W	R/W	PIM2 Address High for BAR1	638
PECFGn_PIM3LAL	0x360	R/W	R/W	PIM3 Address Low for BAR2	638
PECFGn_PIM3LAH	0x364	R/W	R/W	PIM3 Address High for BAR2	639
PECFGn_PIM4LAL	0x368	R/W	R/W	PIM4 Address Low for BAR2	639
PECFGn_PIM4LAH	0x36C	R/W	R/W	PIM4 Address High for BAR2	639
PECFGn_PIM34SAL	0x370	R/W	R/W	PIM34 Size Address Low for BAR2	640
PECFGn_PIM34SAH	0x374	R/W	R/W	PIM34 Size Address High for BAR2	640
PECFGn_PIM5LAL	0x378	R/W	R/W	PIM5 Address Low for BAR3	641
PECFGn_PIM5LAH	0x37C	R/W	R/W	PIM5 Address High for BAR3	641
PECFGn_POM0LAL	0x380	R/W	R/W	POM0 Address Low for BAR0	641
PECFGn_POM0LAH	0x384	R/W	R/W	POM0 Address High for BAR0	642
PECFGn_POM1LAL	0x388	R/W	R/W	POM1 Address Low for BAR1	642
PECFGn_POM1LAH	0x38C	R/W	R/W	POM1 Address High for BAR1	642
PECFGn_POM2LAL	0x390	R/W	R/W	POM2 Address Low for BAR2	643
PECFGn_POM2LAH	0x394	R/W	R/W	POM2 Address High for BAR2	643
PECFGn_POM3LAL	0x398	R/W	R/W	POM3 Address Low for BAR3	643
PECFGn_POM3LAH	0x39C	R/W	R/W	POM3 Address High for BAR3	644
PECFGn_POM4LAL	0x3A0	R/W	R/W	POM4 Address Low for BAR4	644
PECFGn_POM4LAH	0x3A4	R/W	R/W	POM4 Address High for BAR4	644
PECFGn_RTSID	0x3AC	R/W	R/W	Root Port Source ID	645
PECFGn_RTBAR1	0x3B0	R/W	R/W	Root Port BAR1 Base Address	645
PECFGn_RTBAR2L	0x3B8	R/W	R/W	Root Port BAR2 Base Address Low	645
PECFGn_RTBAR2H	0x3BC	R/W	R/W	Root Port BAR2 Base Address High	646
PECFGn_DLLPPA	0x400	R/W	R/W	Data Link Layer Parameters Programming Access	646
PECFGn_ADERC	0x428	R/W	R/W	Advisory Error Reporting Control	646

Inbound address mapping is determined by the PIM (PCI Inbound Map) and PCI Express Address BAR registers. Outbound address mapping is determined by the POM (PCI Outbound Map) and PLB Address BAR registers. The following sections describe the PIM and POM registers.

20.26.3.1 InRegion Valid Programming Access Register (PECFGn_PIMEN)

Figure 20-191. InRegion Valid Programming Access Register (PECFGn_PIMEN)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:4		Reserved	0x000_0000	R	R
3	BAR3EN	InRegion 3 Valid: 1 Allows the PLB Master to decode and translate the PCI address into a PLB address if it falls within the BAR 3 region	0b0	R/W	R/W
2	BAR2EN	InRegion 2 Valid: 1 Allows the PLB Master to decode and translate the PCI address into a PLB address if it falls within the BAR 2 region	0b0	R/W	R/W
1	BAR1EN	InRegion 1 Valid: 1 Allows the PLB Master to decode and translate the PCI address into a PLB address if it falls within the BAR 1 region.	0b0	R/W	R/W
0	BAR0EN	InRegion 0 Valid: 1 Allows the PLB Master to decode and translate the PCI address into a PLB address if it falls within the BAR 0 region	0b0	R/W	R/W

20.26.3.2 PIM0 Address Low for BAR0 Register (PECFGn_PIM0LAL)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-192. PIM0 Address Low for BAR0 Register (PECFGn_PIM0LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:10	LAL	PIM0 Address Low for BAR0: Used to translate the PCI address into a PLB address for transactions that fall within the first address region (PCI BAR0 memory region) and are located below the offset indicated by PECFGn_PIM01SAL) and PECFGn_PIM01SAH	0x000_0000	R/W	R/W
9:0		Reserved	0x000	R	R

20.26.3.3 PIM0 Address High for BAR0 Register (PECFGn_PIM0LAH)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-193. PIM0 Address High for BAR0 Register (PECFGn_PIM0LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	PIM0 Address High for BAR0: Used to translate the PCI address into a PLB address for transactions that fall within the first address region (PCI BAR0 memory region) and are located below the offset indicated by PECFGn_PIM01SAL and PECFGn_PIM01SAH	0x0000_0000	R/W	R/W

20.26.3.4 PIM1 Address Low for BAR0 Register (PECFGn_PIM1LAL)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-194. PIM1 Address Low for BAR0 Register (PECFGn_PIM1LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	LAL	PIM1 Address Low for BAR0: Used to translate the PCI address into a PLB address for transactions that fall within the first address region (PCI BAR0 memory region) and are located below the offset indicated by PECFGn_PIM01SAL and PECFGn_PIM01SAH	0x0000	R/W	R/W
19:0		Reserved	0x0_0000	R	R

20.26.3.5 PIM1 Address High for BAR0 Register (PECFGn_PIM1LAH)

This register enables customizing the PCI Express address space PIM1 within the PLB address space.

Figure 20-195. PIM1 Address High for BAR0 Register (PECFGn_PIM1LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	PIM1 Address High for BAR0: Used to translate the PCI address into a PLB address for transactions that fall within the first address region (PCI BAR0 memory region) and are located above the offset indicated by PECFGn_PIM01SAL and PECFGn_PIM01SAH	0x0000_0000	R/W	R/W

20.26.3.6 PIM01 Size Address Low for BAR0 Register (PECFGn_PIM01SAL)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-196. PIM01 Size Address Low for BAR0 Register (PECFGn_PIM01SAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:10	SAL	PIM01 Size Address Low for BAR0: Specifies the offset that splits the first address region (PCI BAR0 memory region) into two regions. Transactions within the first address region and that are located below the offset indicated by this register are translated using PIM0. All other transactions are translated using PIM1. The value of the offset range from 1KB to the size of the first address region (2^{63}). Bits 1:53 of the offset that splits the 64-bit BAR0 memory region can be masked. The mask defines the bits that are considered by the address decoder, thus determining the value of the region offset. The offset value can range from 1KB (2^{10}) to 2^{63}: 0000000000.....0000000000 - 2^{63}B 1000000000.....0000000000 - 2^{62}B 1100000000.....0000000000 - 2^{61}B --- 1111111111.....1111111100 - 2^{12}B 1111111111.....1111111111 - 2^{11}B 1111111111.....1111111111 - 2^{10}B The offset value must be correlated with the mask input for the BAR0 so that the indicated offset value is not larger than the BAR0 size.	0x00_0000	R/W	R/W
9:0		Reserved	0x000	R	R

20.26.3.7 PIM01 Size Address High for BAR0 Register (PECFGn_PIM01SAH)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-197. PIM01 Size Address High for BAR0 Register (PECFGn_PIM01SAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31		Reserved	0	R	R
30:0	SAH	PIM01 Size Address High for BAR0: Sets the offset that splits the first address region (PCI BAR0 memory region) into two regions. Transactions within the first address region and that are located below the offset value indicated by this register are translated using PIM0. All other transactions are translated using PIM1. The offset value can range from 1KB to the size of the first address region (2^{63}). BAR0 memory region offset Mask bits 1:31. See explanation for PECFGn_PIM01SAL[SAL].	0x0000_0000	R/W	R/W

20.26.3.8 PIM2 Address Low for BAR1 Register (PECFGn_PIM2LAL)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-198. PIM2 Address Low for BAR1 Register (PECFGn_PIM2LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:7	LAL	PIM2 Address Low for BAR1: Used to translate the PCI address into a PLB address for transactions that fall within the second address region (PCI BAR1 I/O region)	0x000_0000	R/W	R/W
6:0		Reserved	0x00	R	R

20.26.3.9 PIM2 Address High for BAR1 Register (PECFGn_PIM2LAH)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-199. PIM2 Address High for BAR1 Register (PECFGn_PIM2LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	PIM2 Address High for BAR1: Used to translate the PCI address into a PLB address for transactions that fall within the second address region (PCI BAR1 I/O region)	0x0000_0000	R/W	R/W

20.26.3.10 PIM3 Address Low for BAR2 Register (PECFGn_PIM3LAL)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-200. PIM3 Address Low for BAR2 Register (PECFGn_PIM3LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:10	LAL	PIM3 Address Low for BAR2: Used to translate the PCI address into a PLB address for transactions that fall within the third address region (PCI BAR2 memory region) and are located below the offset indicated by PECFGn_PIM34SAL and PECFGn_PIM34SAH	0x00_0000	R/W	R/W
9:0		Reserved	0x000	R	R

20.26.3.11 PIM3 Address High for BAR2 Register (PECFGn_PIM3LAH)

This register enables customizing the PCI Express address space PIM3 within the PLB address space.

Figure 20-201. PIM3 Address High for BAR 2 Register (PECFGn_PIM3LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	PIM3 Address High for BAR2: Used to translate the PCI address into a PLB address for transactions that fall within the third address region (PCI BAR2 memory region) and are located below the offset indicated by PECFGn_PIM34SAL and PECFGn_PIM34SAH	0x0000_0000	R/W	R/W

20.26.3.12 PIM4 Address Low for BAR2 Register (PECFGn_PIM4LAL)

This register enables customizing the PCI Express address space PIM4 within the PLB address space.

Figure 20-202. PIM4 Address Low for BAR2 Register (PECFGn_PIM4LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	LAL	PIM4 Address Low for BAR2: Used to translate the PCI address into a PLB address for transactions that fall within the third address region (PCI BAR2 memory region) and are located above the offset indicated by PECFGn_PIM34SAL and PECFGn_PIM34SAH	0x0000	R/W	R/W
19:0		Reserved	0x0_0000	R	R

20.26.3.13 PIM4 Address High for BAR2 Register (PECFGn_PIM4LAH)

This register enables customizing the PCI Express address space PIM4 within the PLB address space.

Figure 20-203. PIM4 Address High for BAR2 Register (PECFGn_PIM4LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	PIM4 Address High for BAR2: Used to translate the PCI address into a PLB address for transactions that fall within the third address region (PCI BAR2 memory region) and are located above the offset indicated by PECFGn_PIM34SAL and PECFGn_PIM34SAH	0x0000_0000	R/W	R/W

20.26.3.14 PIM34 Size Address Low for BAR 2 Register (PECFGn_PIM34SAL)

This register enables customizing the PCI Express address space PIM3 and PIM4 within the PLB address space.

Figure 20-204. PIM34 Size Address Low for BAR2 Register (PECFGn_PIM34SAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:10	SAL	PIM34 Size Address Low for BAR2: Sets the offset that splits the third address region (PCI BAR2: memory region) into two regions. Transactions within the third address region and that are located below the offset indicated by this register are translated using PIM3. All other transactions are translated using PIM4. The value of the offset can vary between 1 KB and the size of the third address region (2^{63}) Bits 1:53 of the offset that splits the 64-bit BAR2 memory region can be masked. The mask defines the bits that are considered by the address decoder, thus determining the value of the region offset. The offset value can range from 1KB (2^{10}) to 2^{63}: 0000000000.....0000000000 - 2^{63}B 1000000000.....0000000000 - 2^{62}B 1100000000.....0000000000 - 2^{61}B --- 1111111111.....1111111100 - 2^{12}B 1111111111.....1111111111 - 2^{11}B 1111111111.....1111111111 - 2^{10}B The offset value must be correlated with the mask input for the BAR2 so that the indicated offset value is not larger than the BAR2 size.	0x00_0000	R/W	R/W
9:0		Reserved	0x000	R	R

20.26.3.15 PIM34 Size Address High for BAR2 Register (PECFGn_PIM34SAH)

This register enables customizing the PCI Express address space PIM within the PLB address space.

Figure 20-205. PIM34 Size Address High for BAR2 Register (PECFGn_PIM34SAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31		Reserved	0	R	R
30:0	SAH	PIM34 Size Address High for BAR2: Sets the offset that splits the third address region (PCI BAR2: memory region) into two regions. Transactions within the third address region and that are located below the offset indicated by this register are translated using PIM 3. All other transactions are translated using PIM4. The value of the offset can vary between 1 KB and the size of the third address region (2^{63}). BAR2 memory region offset Mask bits (1:31). See explanation for PECFGn_PIM34SAL[SAL].	0x0000_0000	R/W	R/W

20.26.3.16 PIM5 Address Low for BAR3 Register (PECFGn_PIM5LAL)

This register enables customizing the translation of the PCI Express address space into the PLB address space.

Figure 20-206. PIM5 Address Low for BAR3 Register (PECFGn_PIM5LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:7	LAL	PIM5 Address Low for BAR3: Used to translate the PCI address into a PLB address for transactions that fall within the fourth address region (PCI BAR3 Expansion ROM region)	0x000_0000	R/W	R/W
6:0		Reserved	0x00	R	R

20.26.3.17 PIM5 Address High for BAR3 Register (PECFGn_PIM5LAH)

This register enables customizing the translation of the PCI Express address space into the PLB address space

Figure 20-207. PIM5 Address High for BAR3 Register (PECFGn_PIM5LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	PIM5 Address High for BAR3: Used to translate the PCI address into a PLB address for transactions that fall within the third address region (PCI BAR3 Expansion ROM region)	0x0000_0000	R/W	R/W

20.26.3.18 POM0 Address Low for BAR0 Register (PECFGn_POM0LAL)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-208. POM0 Address Low for BAR0 Register (PECFGn_POM0LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:27	LAL	POM0 Address Low for BAR0: Used to translate the PLB address into a PCI address for transactions that fall within the first address region (PLB BAR0 memory region)	0x00	R/W	R/W
26:0		Reserved	0x000_0000	R	R

20.26.3.19 POM0 Address High for BAR0 Register (PECFGn_POM0LAH)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-209. POM0 Address High for BAR0 Register (PECFGn_POM0LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	POM0 Address High for BAR0: Used to translate the PLB address into a PCI address for transactions that fall within the first address region (PLB BAR0 memory region)	0x0000_0000	R/W	R/W

20.26.3.20 POM1 Address Low for BAR1 Register (PECFGn_POM1LAL)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-210. POM1 Address Low for BAR1 Register (PECFGn_POM1LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:20	LAL	POM1 Address Low for BAR1: Used to translate the PLB address into a PCI address for transactions that fall within the second address region (PLB BAR1 memory region)	0x000	R/W	R/W
19:0		Reserved	0x0_0000	R	R

20.26.3.21 POM1 Address High for BAR1 Register (PECFGn_POM1LAH)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-211. POM1 Address High for BAR1 Register (PECFGn_POM1LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	POM1 Address High for BAR1: Used to translate the PLB address into a PCI address for transactions that fall within the second address region (PLB BAR1 memory region)	0x0000_0000	R/W	R/W

20.26.3.22 POM2 Address Low for BAR2 Register (PECFGn_POM2LAL)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-212. POM2 Address Low for BAR2 Register (PECFGn_POM2LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:7	LAL	POM2 Address Low for BAR2: Used to translate the PLB address into a PCI address for transactions that fall within the third address region (PLB BAR2 memory or I/O region)	0x000_0000	R/W	R/W
6:0		Reserved	0x00	R	R

20.26.3.23 POM2 Address High for BAR2 Register (PECFGn_POM2LAH)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-213. POM2 Address High for BAR2 Register (PECFGn_POM2LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	POM2 Address High for BAR2: Used to translate the PLB address into a PCI address for transactions that fall within the third address region (PLB BAR2 memory or I/O region)	0x0000_0000	R/W	R/W

20.26.3.24 POM3 Address Low for BAR3 Register (PECFGn_POM3LAL)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-214. POM3 Address Low for BAR3 Register (PECFGn_POM3LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:7	LAL	POM3 Address Low for BAR3: Used to translate the PLB address into a PCI address for transactions that fall within the fourth address region (PLB BAR3 message region)	0x000_0000	R/W	R/W
6:0		Reserved	0x00	R	R

20.26.3.25 POM3 Address High for BAR3 Register (PECFGn_POM3LAH)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-215. POM3 Address High for BAR3 Register (PECFGn_POM3LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	POM3 Address High for BAR3: Used to translate the PLB address into a PCI address for transactions that fall within the fourth address region (PLB BAR3 message region)	0x0000_0000	R/W	R/W

20.26.3.26 POM4 Address Low for BAR4 Register (PECFGn_POM4LAL)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-216. POM4 Address Low for BAR4 Register (PECFGn_POM4LAL)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:7	LAL	POM4 Address Low for BAR4: Used to translate the PLB address into a PCI address for transactions that fall within the fifth address region (PLB BAR4 configuration region)	0x000_0000	R/W	R/W
6:0		Reserved	0x00	R	R

20.26.3.27 POM4 Address High for BAR4 Register (PECFGn_POM4LAH)

This register enables customizing the translation of the PLB address space into the PCI Express address space.

Figure 20-217. POM4 Address High for BAR4 Register (PECFGn_POM4LAH)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	LAH	POM4 Address High for BAR4: Used to translate the PLB address into a PCI address for transactions that fall within the fifth address region (PLB BAR4 configuration region)	0x0000_0000	R/W	R/W

20.26.3.28 Root Port Source ID Register (PECFGn_RTSID)

Figure 20-218. Root Port Source ID Register (PECFGn_RTSID)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:24	BN	Bus Number	0x00	NA	R/W
23:19	DN	Device Number	0b0_0000	NA	R/W
18:16	FN	Function Number	0b000	NA	R/W
15:0		Reserved	0b0000	NA	R

20.26.3.29 Root Port BAR1 Base Address Register (PECFGn_RTBAR1)

This register specifies PCI Express Base Address 1 (BAR1 for inbound transactions of a Root Port).

Figure 20-219. Root Port BAR1 Base Address Register (PECFGn_RTBAR1)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:2	BAR1	Root Port Base Address 1 (32-bit BAR1)	Software Initialization See PECFGn_BAR1MPA[BAR1]	NA	R/W
1		Reserved	0b0	NA	R
0	IO	Space Indicator: I/O	0b1	NA	R

20.26.3.30 Root Port BAR2 Base Address Low Register (PECFGn_RTBAR2L)

This register specifies PCI Express Base Address 2 (BAR2 for inbound transactions of a Root Port).

Functionally, the Root Port BAR2 Base Address Low and High registers can be viewed as a single 64-bit register that defines the PLB Base Address BAR2 for inbound transactions.

Figure 20-220. Root Port BAR2 Base Address Low Register (PECFGn_RTBAR2L)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:4	BAR2	Lower Base Address 2 (64-bit BAR2)	Software Initialization See PECFGn_BAR2LMPA[BAL]	NA	R/W
3	PFECH	Prefetchable	Software Initialization See PECFGn_BAR2LMPA[PEF]	NA	R
2:1	TYPE	Memory Type: 64-bit Access Space	Software Initialization See PECFGn_BAR2LMPA[BAT]	NA	R
0	MS	Space Indicator: Memory	0b0	NA	R

20.26.3.31 Root Port BAR2 Base Address High Register (PECFGn_RTBAR2H)

This register specifies PCI Express Base Address 2 (BAR2 for inbound transactions of a Root Port).

Functionally, the Root Port BAR2 Base Address Low and High registers can be viewed as a single 64-bit register that defines the PLB Base Address BAR2 for inbound transactions.

Figure 20-221. Root Port BAR2 Base Address High Register (PECFGn_RTBAR2H)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:0	BAH	Upper Base Address (64-bit BAR2)	Software Initialization See PECFGn_BAR2HMPA[BAH]	R/W	R/W

20.26.3.32 Data Link Layer Parameters Programming Access Register (PECFGn_DLLPPA)

This register specifies the number of fast training sequences that a Root Port must send to the Endpoint to exit L0 state.

Figure 20-222. Data Link Layer Parameters Programming Access Register (PECFGn_DLLPPA)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:8		Reserved	0x00_0000	RO	RO
7:0	NFTS	System Local NFTS Programming Access: Number of Fast Training Sequences (NFTS) that the Root Port must send to the Endpoint to exit state L0.	0x09	R/W	R/W

20.26.3.33 Advisory Error Reporting Control Register (PECFGn_ADERC)

This register specifies advisory error reporting introduced in *PCI Express Specification* Revision 1.1.

Figure 20-223. Advisory Error Reporting Control Register (PECFGn_ADERC)

Bit(s)	Name	Description	Reset Value	PCI Express Attributes Endpoint Only	PLB Attributes
31:3		Reserved	0x0000_0000	RO	RO
2	TOEARE	TimeOut Error Advisory Reporting Enable: 1 The CFG function reports TimeOut errors as correctable (advisory non fatal) instead of non fatal.			
1	PTEARE	Poisoned TLP Error Advisory Reporting Enable: 1 The CFG function reports Poisoned TLP errors as correctable (advisory non fatal) instead of non fatal.	0b0	R/W	R/W

0	EEARE	ECRC Error Advisory Reporting Enable: 1 The CFG function reports ECRC errors as correctable (advisory non fatal) instead of non fatal.	0b0	R/W	R/W

21. Serial ATA (SATA) (PPC460EX/EXr only)

The Serial Advanced Technology Attachment (ATA) interface provides an interface to physical storage devices. It shares the High-Speed SerDes with the PCI-Express interface with 1-Lane. This interface is available only on the PPC460EX/EXr only.

Features include:

- Compliant with Serial ATA Revision 2.5 Specification
- Supports SATA 1.5Gbps Generation 1 and 3 Gbps Generation 2 speeds
- Supports device hot-plugging
- Supports power management
- Supports BIST loopback modes
- Dedicated DMA controller support to optimize performance and off load CPU
- Separate 512B transmit and receive buffers
- ATA/ATAPI master-only emulation
- DMA flow control and interrupt side-band signals
- Supports Port Multiplier with command-based switching

21.1 General Product Description

The PPC460EX/EXr implements the Serial Advanced Technology Attachment (ATA) storage interface for physical storage devices.

Figure 21-1. Typical SATA Host

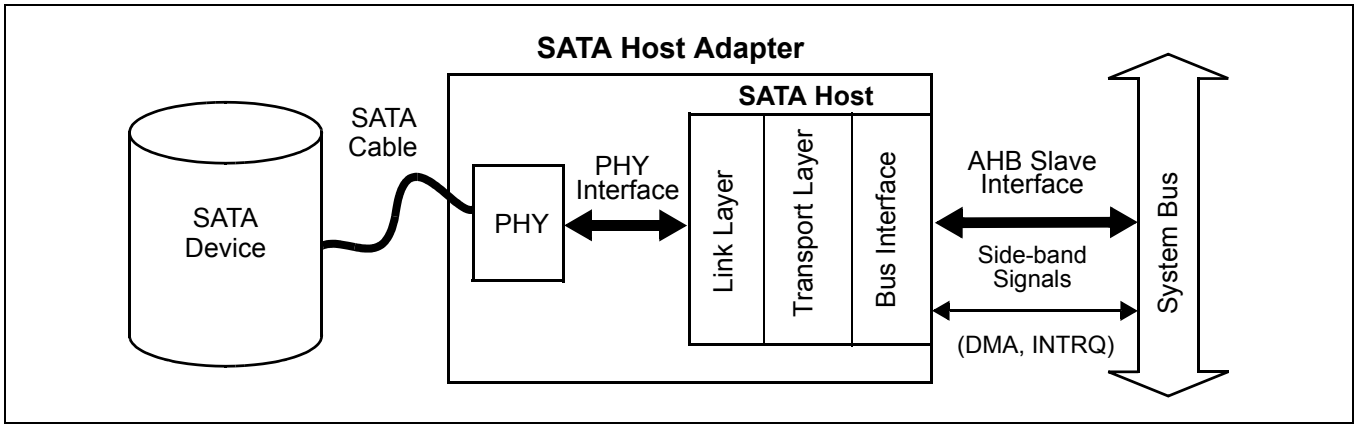


Figure 21-2. Host Block Diagram

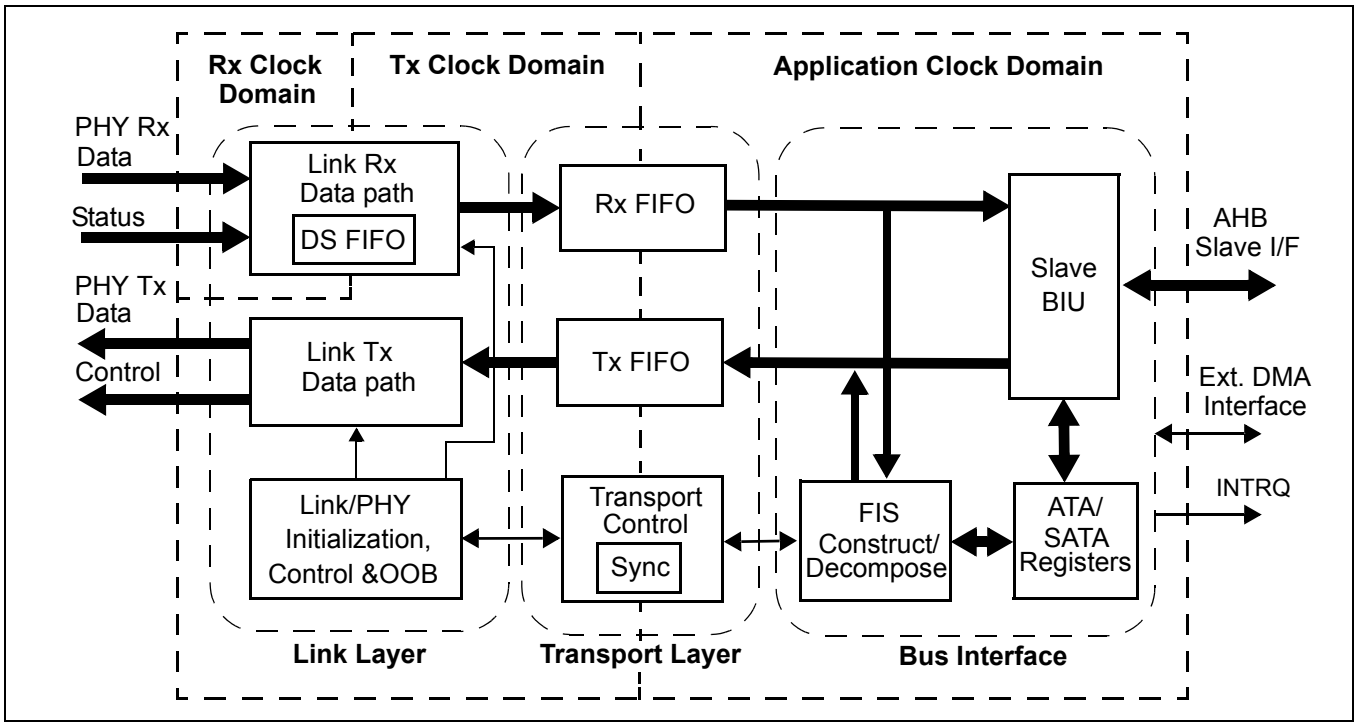
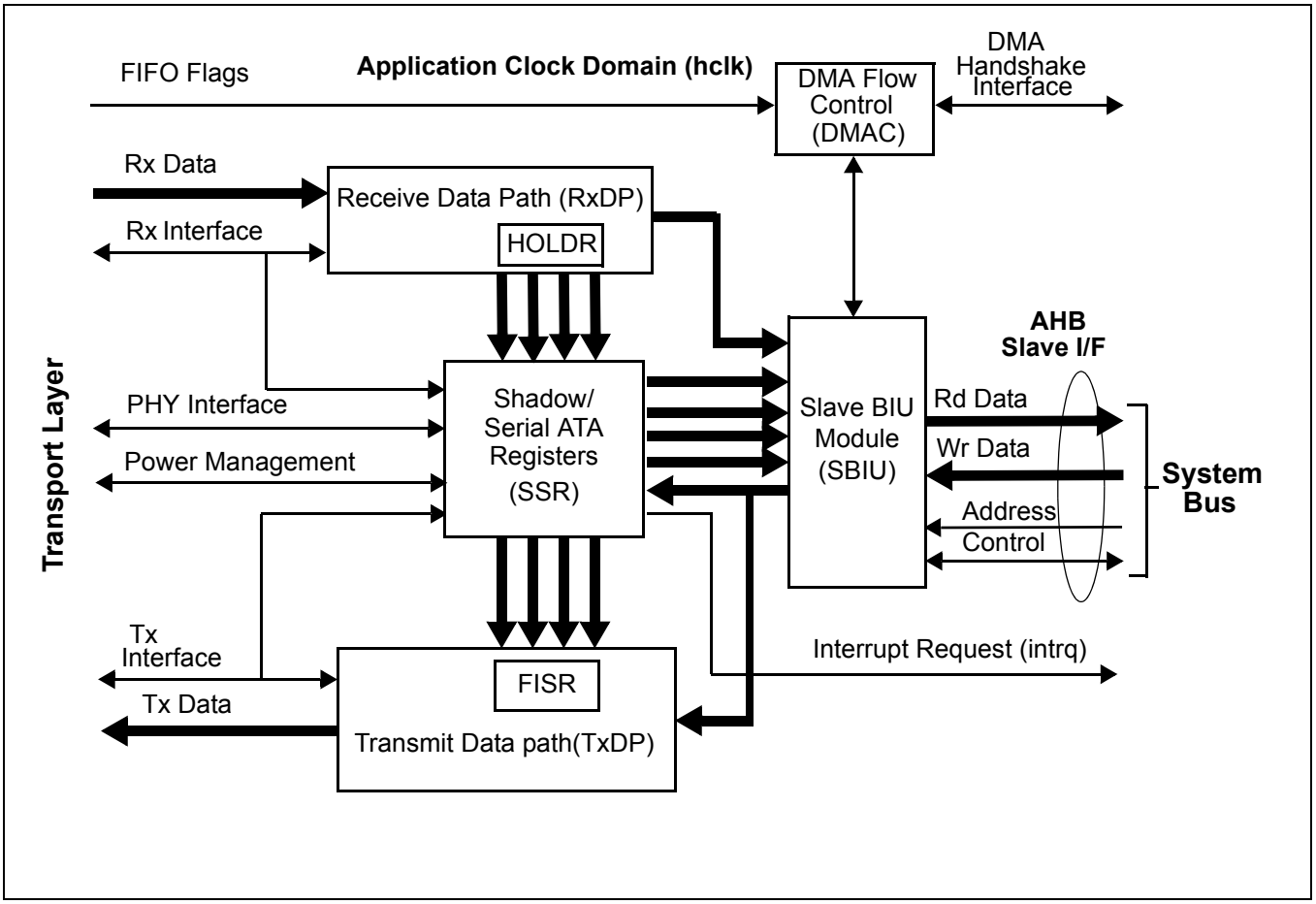


Figure 21-3. Bus Interface Block Diagram



21.2 Programmed I/O Operation

The following sections describe the programmed I/O (PIO) operation.

21.2.1 Bus Interface FIS Construction

Any of the following conditions initiate Frame Information Structure (FIS) construction:

- ATA Command or Device Control register write (Register FIS)
- ATA Data register write and previous FIS was PIO Setup FIS (PIO Data FIS)
- DMA Activate FIS or DMA Setup FIS with A = 1 bit is received (DMA Data FIS)
- BIST Control register is written (BIST Activate FIS)

The corresponding FIS is constructed in the TxDP module and written (pushed) into the TxFIFO for transmission by the Link Layer.

Register FIS (Host to Device) is used by the Host to send a command to the device or initiate a control function (for example, Device Reset). Register FIS is constructed in the TxDP FISR registers by transferring ATA Shadow register content to FISR upon application software write to either ATA Command or Control register.

21.2.2 Read Transfer (Device to Host)

PIO read transfer starts with the host issuing a "PIO Read" command to the device by sending a corresponding Register FIS. After the command has been processed, the device sends the PIO Setup FIS to the host with D bit = 1. The content of the FIS is loaded into the HOLDR registers. The device sends the PIO Data FIS. When the header of the PIO Data FIS has been received, intrq is asserted. The TSM detects Data FIS in the RxFIFO and signals the Bus Interface to transfer "Initial" Error and Status register content from the HOLDR to the ATA registers when the Data FIS is being received. RxFIFO is logically mapped to the 16-bit ATA Data register (SATA0_CDR0). The data DWORD is popped out of the RxFIFO every two read accesses to this register. The Bus Interface keeps track of the read accesses and signals a "PIO count reached" condition, indicating the end of the current PIO transfer to the Transport Layer. The transfer "Ending" Status register content is then transferred from the HOLDR registers to the ATA Status register.

If the device has more PIO data to send, the above process is repeated from the device sending PIO Setup FIS followed by the Data FIS until all data has been transferred.

21.2.3 Write Transfer (Host to Device)

A PIO write transfer is similar to the PIO read transfer, except D = 0 in the PIO Setup FIS and the "Initial" Error and Status register content is transferred to the Shadow registers immediately upon successful reception of the PIO Setup FIS. TxFIFO is logically mapped to the 16-bit ATA Shadow Data register (SATA0_CDR0). The data DWORD is pushed into the TxFIFO every two application software writes to this register. The Bus Interface keeps track of the write accesses and signals a "PIO count reached" condition indicating the end of the current PIO transfer to the Transport Layer. The transfer "Ending" Status register content is transferred from the HOLDR registers to the ATA Status register.

If the transfer is not finished, the above process is repeated from the device sending the PIO Setup FIS, followed by the host sending the Data FIS until all data has been transferred. The device then sends Register FIS with the ending command status.

21.2.4 Slow Device Access

In some cases a device might transfer PIO data at a very slow rate (for example, ATA to SATA bridge sending one DWORD at a time at a 4MB/s rate). The SATA Host AHB slave interface inserts wait states (hready = 0) into the PIO access until either the RxFIFO contains data, or the TxFIFO contains free space.

If the device does not supply data during a PIO read operation or accept data during a PIO write operation within 256 hclk cycles, the SATA Host generates an ERROR response on the hresp output to prevent a possible bus lock-up condition from occurring. This delay corresponds to about a 2.3 MB/s transfer rate with a 300MHz hclk, assuming back-to-back software reads (300MWORD/s divided by 256 = 1.17MWORD/s, or 2.34MB/s).

21.3 Direct Memory Access Operation

The SATA interface has a dedicated DMA controller to support DMA data transfers and provide several handshaking signals for flow control.

21.3.1 DMA Initialization

The DMA initialization process is described as follows:

1. The host software programs (initializes and enables) the DMA controller with all the parameters necessary for a given transfer (for example, transfer direction, address pointers, burst size.).

2. The host software initializes corresponding MRD/MWR fields of the SATA0_DBTSR register, then sets RXCHEN/ TXCHEN bits of the SATA0_DMACR register depending on the transfer direction. The SATA0_DMACR.TXMODE bit selects whether Tx Data FIS is closed after dma_finish_tx is asserted (TXMODE = 0) or after SATA0_DMACR.TXCHEN is cleared (TXMODE = 1).
3. The host software issues a DMA command to the device by writing to the Command register resulting in the Register FIS transmission to the device.
4. The device executes the command and notifies the host that it is ready to send/receive data. During a read operation the device sends the Data FIS to the host; during a write, a DMA Activate or a DMA Setup FIS with A = 1 requesting Data FIS from the host is sent.
5. The data is then transferred between the two DMA agents: system memory and SATA Host peripheral (both system bus slaves). DMA controller acts as a flow controller.

21.3.2 DMA Read Transfer (Device to Host)

During a read operation the data is transferred from the RxFIFO to the system memory through the DMA channel. RxFIFO acts as a source of data (source peripheral) and system memory, as a destination of data (destination memory). Data is received in the RxFIFO from the device in the form of one or more Data FISs. The Transport Layer decodes the FIS type and initiates a DMA transfer. The Bus Interface generates DMA requests to the DMA controller (dma_req_rx or dma_single_rx) based on the RxFIFO "empty" and "almost empty" flags.

The following sequence of events occurs during a DMA read transfer:

1. The Transport Layer detects a Data FIS in the RxFIFO and notifies the Bus Interface to start DMA transfer;
2. The Bus Interface asserts the dma_req_rx signal to initiate a burst read transaction if RxFIFO has enough data for at least one burst or dma_single_rx to initiate a single transaction if RxFIFO has at least one data DWORD. MRD field of the DBTSR register should be set to the burst transaction size expected on the bus in DWORDs prior to setting RXCHEN bit of the SATA0_DMACR register.
3. The DMA controller initiates single or burst read cycles on the system bus to the SATA0_DMADR location to transfer data from the RxFIFO to its internal FIFO and then to destination memory.
4. The DMA controller asserts dma_ack_rx to indicate the DMA transaction completion. The Bus Interface detects dma_ack_rx assertion and negates dma_req_rx and dma_single_rx.
5. The process is repeated until all the data in the Data FIS has been transferred. The Bus Interface suspends the transfer by negating dma_req_rx or dma_single_rx. The device either sends another Data FIS or Register FIS with command end-status information if all data was transferred. Software clears the RXCHEN bit of the SATA0_DMACR register.

21.3.3 DMA Write Transfer (Host to Device)

During a write operation the data is transferred from the system memory to the TxFIFO through the DMA channel. TxFIFO acts as a data destination (destination peripheral) and system memory acts as a source of data (source memory). The Bus Interface generates DMA requests to the DMA controller (dma_req_tx or dma_single_tx) based on the TxFIFO "full" and "almost full" flags.

The following sequence of events occurs during a DMA write transfer:

1. The device signals its readiness to receive data by sending a DMA activate or DMA Setup FIS with A = 1 to the host. The Transport Layer detects this FIS and notifies the Bus Interface to start a DMA transfer.
2. The Bus Interface asserts dma_req_tx to initiate a burst write transaction if TxFIFO has enough space for at least one burst or dma_single_tx if TxFIFO has space for at least one DWORD. MWR field of the SATA0_DBTSR register should be set to the burst transaction size expected on the bus in DWORDs prior to setting TXCHEN bit of the SATA0_DMACR register.

3. The DMA controller generates single or burst write cycles on the system bus to the DMADR location to transfer data from the system memory to the TxFIFO.
4. The DMA controller asserts dma_ack_tx to indicate the DMA transaction completion. The Bus Interface detects dma_ack_tx assertion and negates dma_req_tx and dma_single_tx.
5. The process is repeated until the number of bytes transferred reaches 8192 bytes (2048 DWORDs) or the DMA controller asserts dma_finish_tx (same timing as dma_ack_tx) indicating the completion of the current block (if SATA0_DMACR.TXMODE = 0). The Bus Interface notifies the Link Layer to close the FIS (generate CRC and EOFp) by setting bit 32 of the last DWORD. The DMA controller should be suspended at this time. A device sends either a DMA Activate FIS to request another Data FIS or a Register FIS with command end-status information. When all data has been transferred, the software clears the TXCHEN bit of the SATA0_DMACR register.

21.3.4 DMA Requirements

DMA block and burst transaction sizes are critical for proper DMA operation. These sizes should be selected properly to ensure error-free bus transfers. It is required that the DMA write burst transfer does not cross the 8192-B Data FIS boundary, because the Transport Layer maintains the DMA state for the duration of the Data FIS transmission.

Note: Violation of these requirements results in transfer abort and bus ERROR response.

SATA Host DMA operation requirements are as follows:

- DMA controller block/burst sizes should be set as follows:
  - Receive (read) operation: The block can be any size up to the DMA transfer size. Burst can be any size up to the maximum limited by the SATA0_DBTSR.MRD value.
  - Transmit (write) operation: Block and burst sizes must be selected so that the burst does not cross a 8192-byte FIS boundary.

    If the block size is set to be exactly 8192B (except for the last one), then the burst size can be any value limited by SATA0_DBTSR.MWR.

    If the block size is larger than 8192B (for example, 16 KB, 32 KB), it must be an integer multiple of 8192 and the burst size must be a 2^n number of DWORDs (n = 0 to 10, limited by SATA0_DBTSR.MWR).

    If the block size is smaller than 8192B, then the several consecutive block sizes must add up to 8192 bytes and the burst size can be any value limited by SATA0_DBTSR.MWR. For example: 1KB - 4KB - 2KB - 1KB, 1KB - 2KB - 2KB - 3KB.

    Note: Using SATA0_DMACR.TXMODE = 0 in this case results in transmission of the Data FIS smaller than 8192 B, which is violation of the SATA spec. It is recommended using SATA0_DMACR.TXMODE = 1 for any block/burst size.
  - Transmit (write)/Receive (read) operation: The block size must be 32-bit aligned.
- The SATA0_DBTSR register should be programmed with the burst transaction size values (in DWORDs): MRD field for read transfer, MWR field for write transfer prior to enabling the channel, and the same values should be used to program DMA controller.
- The SATA Host should be notified that the DMA channel is enabled by setting the corresponding TX/RXCHEN bit of the SATA0_DMACR register prior to issuing a DMA command to the device. The TX/RXCHEN bits must be cleared by software after the DMA controller has transferred all data for the command. These bits are cleared automatically on completion of the first-party DMA data phase.

- DMA transaction transfer size should be: 16-bits for 16-bit data bus, 32-bits for a 32-bit or wider data bus. The address should be 16- or 32-bit-aligned for a 16-bit bus, and 32-bit aligned for a 32-bit and wider bus, and be within the SATA0_DMADR range.

21.3.5 DMA Termination

A DMA transfer can be terminated (aborted) before its normal completion either by a host or a device in the following cases:

- DMA Read Transfer Abort
- DMA Write Transfer Abort

21.3.5.1 DMA Read Transfer Abort

A DMA read transfer is aborted:

- By the device when it stops sending Data FIS and sends Register FIS with command end status. Host software detects this condition with ATA Status/Error registers and disables the DMA controller.
- By the host software when it disables the DMA channel by clearing the RXCHEN bit of the SATA0_DMACR register. This is usually done prior to resetting the device with SRST/Device Reset command (see next case).
- By the host software when it sets SRST bit of the Control register or issues Device Reset command to the device. The Transport Layer notifies the Link Layer of the condition. The current Data FIS is "flushed" from the RxFIFO and the corresponding Register FIS is sent to the device. Host software disables the channel and clears RXCHEN/TXCHEN bits of the SATA0_DMACR register.

21.3.5.2 DMA Write Transfer Abort

A DMA write transfer is aborted:

- By the device when it sends DMATp to the host requesting to abort current transfer. The Transport Layer detects this condition and sets DMAT bit of the SATA0_INTPR register. Interrupt output intrq is asserted if the DMATM bit of the SATA0_INTMR register is set.

  Note: The SATA specification recommends ignoring reception of the DMATp and completing the current transfer for better compatibility.
- By the host software when it disables (deactivates) the channel by clearing the TXCHEN bit of the SATA0_DMACR register. The current Data FIS is closed. Host software issues a SRST/Device Reset command to the device (see next case).
- By the host software when it sets SRST bit of the register or issues Device Reset command to the device. The Transport Layer notifies the Link Layer of the condition. The current transfer is interrupted, TxFIFO is "flushed" and the corresponding Register FIS is sent to the device. Host software is responsible for disabling the DMA channel in this case.

21.3.6 First-Party DMA

First-Party DMA is a method of implementing the native SATA command queueing mechanism. The details can be found in the SATA specification. The SATA Host provides several registers to implement this functionality, which serve as hooks for software. The functionality is the same as described in the *Direct Memory Access Operation* on page 652, except it is preceded by the following:

1. The host software sets SActive register bit corresponding to the command TAG value and issues "First-Party DMA" command. Register FIS is sent to the device and BSY is set in the Status register.

2. Device adds the command to it's internal queue and sends Register FIS to clear BSY bit.
3. Device executes a command from the queue and sends DMA Setup FIS to the host to initiate data transfer phase.

Note: Host software is not allowed to issue a new command to the device during the current First-Party DMA data phase, however, since the BSY bit is cleared, potentially it can. In this case, the SATA Host sets the BSY bit and delays the corresponding Register FIS transmission until after the current First-Party DMA data phase is complete, that is, all the data for the current DMA Setup FIS has been transferred.

4. Upon reception of the DMA Setup FIS, the Bus Interface updates corresponding the SATA0_FPTAGR, SATA0_FPBOR, and SATA0_FPTCR registers from the values in the FIS and sets NEWFP bit of the SATA0_INTPR register. Interrupt output intrq is asserted if NEWFPM bit of the SATA0_INTMR register is set.
5. The host software updates the DMA controller context accordingly and enables the channel for transfer. The host software sets the corresponding TXCHEN or RXCHEN bit in the SATA0_DMACR register.
6. SATA Host generates DMA request signals to the DMA controller to initiate the transfer. The operation proceeds as described in the DMA operations chapter.
7. When the device completes the command data phase, it sends Set Device Bits FIS to the host with the bit corresponding to the TAG value set in the SActive field, so this bit is cleared in the host SActive register.
8. The RXCHEN/TXCHEN bits of the SATA0_DMACR register are cleared automatically by the SATA Host when the current data phase is complete, that is, when the transfer count for the DMA Setup FIS is reached.

21.4 Interrupt Control

The SATA Host provides an interrupt request output intrq to implement system interrupt. It is asserted if any of the interrupt events are set in the SATA0_INTPR register and the corresponding mask bit is set in the SATA0_INTMR or the IPF flag is set and the nIEN bit is cleared in the Device Control register (SATA0_CLR0).

The ATA interrupt pending flag (IPF) is set when the device signals an interrupt condition to the host by sending a Register, Set Device Bits, or PIO Setup FIS with Interrupt bit set (I = 1). The IPF is cleared when the ATA status register (SATA0_CDR7) is read. If IPF is set and nIEN in the Device Control register is cleared (nIEN = 0), SATA Host asserts the external interrupt output intrq.

Note: IPF is an internal register; however, software can check its state by using the SATA0_INTPR.IPF bit. A read access to the SATA0_INTPR.IPF location does not change the IPF register's state.

The intrq output is also asserted if any of the bits in the SATA0_INTPR is set and the corresponding bit in the SATA0_INTMR is set (interrupt event is detected and enabled/unmasked). The following events cause SATA0_INTPR bits to be set:

- DMAT: The DMATp is received from the device during DMA Data FIS transmission.
- NEWFP: A new DMA Setup FIS is received from the device.
- PMABORT: A power mode abort condition is detected by the Link Layer.
- ERR: Any of the SError register bits is set.
- NEWBIST: The new BIST Activate FIS is received from the device
- PRIMERR: The Link Layer detects an error in the primitive DWORD.
- CMD_ABORT: A Register or Set Device Bits FIS is received with Status.ERR = 1 (Command abort status).
- CMD_GOOD: A Register or Set Device Bits FIS is received with Status.ERR = 0 (Command good status).

The SATA0_ERRMR register is used to mask corresponding bits of the SError register before they cause setting the SATA0_INTPR register ERR bit.

21.5 Register Access

The SATA Host occupies 2048 bytes of the system memory space and uses the lower 256 bytes for Shadow, SATA, and SATA Host-specific registers. The rest of the locations (addresses from 0x100 to 0x3FF) are not implemented and return an ERROR response if accessed. All SATA Host register locations are 32-bit aligned.

Address SATA0_CDR0 is used for data transfer to/from the Tx/RxFIFO in PIO mode and SATA0_DMADR in DMA mode. In PIO mode, SATA Host supports only 16-bit single bus transfers. In DMA mode, SATA Host supports 16- or 32-bit single/burst bus transfers.

The ATA/ATAPI Shadow and SATA registers' access is implemented according to the SATA specification. SATA Host-specific registers provide various functions such as: DMA support, interrupt control, testing, PHY/Link control, and so on.

21.6 AHB Error Conditions

The SATA Host AHB slave interface detects a number of illegal conditions during bus transfer, sets SError register bit 20 (ERR_E), and generates ERROR response on the hresp output. Following is the list of these conditions:

- Address is not 32-bit aligned when accessing SATA0_CDR0–SATA0_CDR7, SATA0_CLR0, SATA0_DMADR locations.
- Write access to the read only locations (for example, SATA0_SCR0, SATA0_FPTR, SATA0_FPBOR, SATA0_FPTCR.)
- Address selects non-implemented area of the address space.
- Shadow ATA registers write access when either BSY or DRQ bit is set in Status register.
- Transfer size exceeds 32 bits.
- Access to the FIFO (SATA0_CDR0/SATA0_DMADR) and the SATA Host not in PIO or DMA mode.
- Burst access to the FIFO (SATA0_CDR0) in PIO mode or to registers if the burst size exceeds one (burst size of one is acceptable).
- Access to the FIFO (SATA0_CDR0/SATA0_DMADR) in PIO or DMA mode when the RxFIFO is empty, or the TxFIFO is full. This error might result from the AHB burst size exceeding the value programmed in the SATA0_DBTSR register that sets the Rx/TxFIFO threshold values for DMA flow control. DMA bus transfer starts only when there is enough data in the RxFIFO or enough space in the TxFIFO for the requested DMA transaction. In PIO mode, the error is generated if the device supplies read data (or accepts write data) at a rate slower than 2.3MB/s (hclk = 300MHz).

  Note: In some cases, SError bit 20 (SCR1.ERR_E) can be set while the AHB ERROR response is not generated. For example, when a TxFIFO overflow condition is detected at the end of the burst transfer as a result of the last DWORD being written to the TxFIFO, when the FIFO is already full.
- DMA transaction transfer size is 16 bits.

All of the cases above are indications of a system design or operation error and are not expected to happen during normal operation.

21.7 Bus Interface Power Management

The host software can request either PARTIAL or SLUMBER power management states by writing to the SPM field of the SControl (SATA0_SCR2) register (bits16:19). The device requests power management state by transmitting PMREQ_Pp or PMREQ_Sp primitives to the host. The power state machine is implemented in Link Layer power management. It asserts a corresponding signal (phy_partial or phy_slumber) to the PHY to enter the power management state.

Note: Both receive and transmit clocks must be running in either PARTIAL or SLUMBER power management state when phy_partial or phy_slumber output is asserted.

The host can disable transition to power management states when the device requests it using the IPM field of the SControl register (bits 20:23).

The power management state is terminated when either one of the following conditions becomes true:
- Host software issues a new command by writing to the Command register
- Host software requests device reset by toggling SRST bit of the Device Control (SATA0_CLR0) register
- Host software requests BIST mode by writing to the SATA0_TXBISTPD register
- Host software requests transition to active mode by writing to the SControl SPM field (bits 16:19 = 0b0100)
- Device requests interface wake up by transmitting COMWAKE OOB sequence

The state of the interface (active, PARTIAL or SLUMBER power management) is reflected in the IPM field of the SStatus register (bits 20:23).

21.8 Hot-Plug

The SATA Host supports hot-plug through the use of the SError bit 5 (SATA0_SCR1.DIAG_X). It is set every time the Link detects a COMINIT sequence, indicating a new device insertion event or system power-up.

21.9 Reset Conditions

The different reset conditions are identified as follows:

- Power-on reset. This reset is initiated by the system bus immediately after power-on or when it crashes. It is provided by the system bus as an asynchronous, active-low signal. All components of the SATA Host are initialized, including Link Layer, Transport Layer, FIFOs, ATA/SATA registers.
- Hard interface reset (COMRESET). This reset is initiated by the host software by setting bit 31 of the SControl register (DET field). The actions are equivalent to power-on reset. The Link Layer LFSR registers are initialized to the required values.
- Soft interface reset. This reset is initiated by the host software either by setting the SRST bit in the Shadow Device Control register or by issuing an ATAPI DEVICE RESET command (0x08). In both cases, the Register FIS is sent to the device. It is used by the host to reset the device. Any on-going interface activity is interrupted to allow for the Register FIS to be sent to the device.

21.10 BIST Operation

Note: Scrambler/Descrambler is bypassed (disabled) in the SATA Host Link Layer in all BIST modes except when in loopback device far-end retimed mode.

This section discusses the following topics:

- Loopback Device
- Loopback Initiator

21.10.1 Loopback Device

The SATA Host enters one of the BIST loopback device modes when a corresponding BIST Activate FIS is successfully received from the device and is supported by SATA Host.

Upon reception of the valid BIST Activate FIS, the SATA Host sets the NEWBIST bit of the SATA0_INTPR register and asserts interrupt output intrq if NEWBISTM = 1 in the SATA0_INTMR register. SStatus register DET field (bits 28:31) returns 0b0100 value when read.

Note: If the device sends a BIST Activate FIS with a request to enter a non-supported loopback mode, SATA Host responds with R_ERRp response upon reception of the FIS.

The following loopback device modes are supported by SATA Host:

- Far-end retimed
  - The SATA Host receives BIST Activate FIS with Pattern Definition field = 0x10 from the RxFIFO and stores it in the RXBISTPD register.
  - All the data received from the device in the form of a SATA-compliant pattern is retimed in the Link Layer and transmitted back to the device.
- Far-end analog (Host PHY must support this mode)
  - The SATA Host receives BIST Activate FIS with Pattern Definition field = 0x08 from the RxFIFO and stores it in the SATA0_RXBISTPD register.
  - The SATA Host asserts phy_farafelb signal to the PHY to put it to the Far-end analog loopback mode. The PHY receives and retransmits the raw data without retiming.
- Far-end transmit only
  - The SATA Host receives BIST Activate FIS with Pattern Definition field = 0x80 or 0xA0 (scrambling is bypassed) from the RxFIFO and stores it in the SATA0_RXBISTPD register. The following two data DWORDs are stored in the SATA0_RXBISTD1 and SATA0_RXBISTD2 registers.
  - The SATA Host transmits a corresponding SATA non-compliant test pattern to the device based on the value in the SATA0_RXBISTD1 register (bits 0:31):
    - 0xEE27FEF1: Low transition density pattern
    - 0xB5B5B5B5: High transition density pattern
    - 0xABABABAB: Low frequency spectral component pattern
    - 0x7F7F7F7F: Simultaneous switching outputs pattern

Loopback device BIST modes can be exited either when the device signals COMINIT OOB condition, or when the host software sets SControl31 = 1 (COMRESET).

21.10.2 Loopback Initiator

The host software should reset the interface by toggling SControl31 (COMRESET) and wait for the BSY bit to be cleared prior to entering one of the initiator loopback modes (or changing from one BIST mode to another) to ensure the device is in the known state and the SATA Host BIST logic is properly initialized.

The SATA Host enters one of the BIST loopback initiator modes after the host software writes to the SATA0_TXBISTPD and SATA0_BISTCR registers. Any write to the SATA0_TXBISTPD register initiates transmission of the BIST Activate FIS to the device. After the host successfully transmits this FIS, it enters this mode and generates/receives a compliant test pattern. SStatus register DET field (bits 3:0) returns 0b0100 value when read. SATA0_BISTSR and SATA0_BISTFCTR are updated with error/FIS count information for each received BIST FIS.

Note: The device must support either PARTIAL or SLUMBER power modes for near-end analog loopback mode, otherwise it should be initiated with the device disconnected from the host PHY. It is not clear how the device responds if it does not support the requested BIST mode—with R_OKp and ignoring the request or with R_ERRp. In the former case, the host assumes the device has entered BIST mode and starts the test that fails.

The following BIST initiator modes can be requested by the host software:

- Far-end retimed
  - The host software writes the SATA0_BISTCR register PATTERN field to select one of the SATA-defined compliant patterns:
    - 0b000: Simultaneous switching bit pattern
    - 0b001: High transition density bit pattern
    - 0b010: Low transition density bit pattern
    - 0b011: Low frequency spectral component bit pattern
    - 0b100: Composite pattern
  - The host software writes 0x10 to bits 24:31 of the SATA0_TXBISTPD register. Corresponding BIST Activate FIS is sent to the device with the Pattern Definition field (bits 8:15 of the first DWORD) containing this value.
  - After successful transmission of the BIST Activate FIS (the device acknowledges the FIS with R_OKp). The SATA Host continuously generates the requested compliant pattern in the form of BIST frames, and checks for errors on the receive side.
  - SATA0_BISTFCTR register is updated with the received BIST frame count and SATA0_BISTSR is updated with a frame/burst error count. SATA0_BISTDECR is updated with DWORD error count. SError register is updated with CRC, disparity and 10B8B errors for each frame. SATA0_BISTFCTR, SATA0_BISTSR, and SATA0_BISTDECR registers can be cleared by writing to the SATA0_BISTCR with the CNTCLR bit set.
  - To change the pattern, the software writes to the SATA0_BISTCR PATTERN field to select a new pattern.
- Far-end analog
  - The host software requests this mode by writing 0x8 to bits 25:31 of the SATA0_TXBISTPD register. Corresponding BIST Activate FIS is sent to the device with the Pattern Definition field (bits 8:15) of the first DWORD containing this value.
  - The operation proceeds as described in the far-end retimed test above.
- Near-end analog

This mode can be initiated either in one of the power management modes (PARTIAL or SLUMBER) or with the device disconnected from the host PHY (Link NOCOMM state). The host software issues a PARTIAL or SLUMBER power mode request to the device via SControl16:19 and sets the NEALB bit of the SATA0_BISTCR register. The PATTERN field of the SATA0_BISTCR register selects the required BIST pattern.

The SATA Host asserts phy_nearafelb to the PHY. The PHY loops the data from its transmitter to its receiver and ignores any data coming from the device.

The operation proceeds as described in the far-end retimed test above, except BIST Activate FIS is not sent to the device.

- Far-end transmit only
  - The host software writes the required pattern DWORDs to the SATA0_TXBISTD1 and SATA0_TXBISTD2 registers. The host software initiates transmission of the BIST Activate FIS to the device by writing bits 24:31 of the SATA0_TXBISTPD register with the value corresponding to the required mode: Bit 24 is set, bits 27, 28, 29, and 30 cleared, and bits 25, 26, and 29 are used to enable the following options:
    - Bit 26 is set: Bypass ALIGN
    - Bit 26 is set: Bypass scrambling
    - Bit 29 is set: Primitive bit (refer to the SATA specification for more details)
  - The BIST Activate FIS is sent to the device with the Pattern Definition field (bits 8:15) of the first DWORD containing this value.
  - After the device acknowledges the reception of this FIS with R_OKp, the SATA Host disables the PHY receiver and transmitter (any received data is ignored by the Link Layer, transmitter is idle and maintains common mode bias per SATA 1.0a specification).

Loopback initiator BIST modes can be terminated either by the device when it signals COMINIT OOB condition (except the near-end analog mode), or by the host software when it sets SControl31 = 1 (COMRESET).

Note: When BIST patterns are generated by the SATA Host as an initiator, BIST patterns with different character sequences depending on starting running disparity, have a 50% chance of being the desired bit pattern on the high-speed differential pair. This is due to the fact that only one set of data values is used to generate the BIST pattern, regardless of current running disparity at the 8b/10b encoder.

21.11 SATA Host Controller Register Summary

The following table provides a summary of the SATA Host Controller registers along with a cross-reference to the location of the register details.

In addition, see SDR0_SATA_CFC in *AHB Subsystem SDR Registers* on page 136.

The starting memory location for these registers is 0x4 BFFD 1000.

Table 21-1. SATA Host Controller Register Summary

Mnemonic	Register	Address Offset	Access	Page
Shadow ATA/ATAPI				
SATA0_CDR0	Data register in PIO mode **Reset Value:** None Dependencies: Read only for PIO read/receive operation, write-only for PIO write/transmit operation	0x00	R/W	667
SATA0_CDR1	Error register **Reset Value:** 0xFF Feature register (current value) **Reset Value:** 0x00 Feature expanded register (previous value) **Reset Value:** 0x00	0x04	R W W	667

Table 21-1. SATA Host Controller Register Summary (Continued)

Mnemonic	Register	Address Offset	Access	Page
SATA0_CDR2	Sector count register (current value) **Reset Value:** 0xFF Sector count expanded register (previous value) **Reset Value:** 0xFF	0x08	R/W	668
SATA0_CDR3	Sector number register (current value) **Reset Value:** 0xFF Sector number expanded register (previous value) **Reset Value:** 0xFF	0x0C	R/W	668
SATA0_CDR4	Cylinder low register (current value) **Reset Value:** 0xFF Cylinder low expanded register (previous value) **Reset Value:** 0xFF	0x10	R/W	669
SATA0_CDR5	Cylinder high register (current value) **Reset Value:** 0xFF Cylinder high expanded register (previous value) **Reset Value:** 0xFF	0x14	R/W	669
SATA0_CDR6	Device/ Head register **Reset Value:** 0xEF	0x18	R/W	670
SATA0_CDR7	Status register **Reset Value:** 0x7F **Dependencies:** Value is 0x7F on power-up, then 0x80 when device presence is detected via PHY READY condition. Command register **Reset Value:** 0x00	0x1C	R W	670
SATA0_CLR0	Alternative status register **Reset Value:** 0x7F **Dependencies:** Value is 0x7F on power-up, then 0x80 when device presence is detected via PHY READY condition. Device control register **Reset Value:** 0x00	0x20	R W	672
Serial ATA				
SATA0_SCR0	SStatus register **Reset Value:** 0x0000_0000	0x24	R	673
SATA0_SCR1	SError register **Reset Value:** 0x0000_0000	0x28	R/W	674
SATA0_SCR2	SControl register **Reset Value:** 0x0000_0000	0x2C	R/W	676
SATA0_SCR3	SActive register **Reset Value:** 0x0000_0000	0x30	R/W	678
SATA0_SCR4	SNotification register **Reset Value:** 0x0000_0000	0x34	R/W	678
SATA0_SCR5–SATA0_SCR15	Reserved for SATA **Reset Value:** 0x0000_0000 **Dependencies:** Reads to these locations return zeros; writes have no effect.	0x38–0x60	See register	679
SATA Host-Specific: DMA				

Table 21-1. SATA Host Controller Register Summary (Continued)

Mnemonic	Register	Address Offset	Access	Page
SATA0_FPTAGR	First Party DMA tag register **Reset Value:** 0x0000_0000	0x64	R	679
SATA0_FPBOR	First Party DMA buffer offset register **Reset Value:** 0x0000_0000	0x68	R	679
SATA0_FPTCR	First Party DMA transfer count register **Reset Value:** 0x0000_0000	0x6C	R	679
SATA0_DMACR	DMA Control Register **Reset Value:**	0x70	R/W	680
SATA0_DBTSR	DMA Burst Transaction Size Register **Reset Value:** 0x0014_0010	0x74	R/W	680
SATA Host-Specific: Interrupt				
SATA0_INTPR	Interrupt Pending Register **Reset Value:** 0x0000_0000	0x78	R/W	681
SATA0_INTMR	Interrupt Mask Register **Reset Value:** 0x0000_0000	0x7C	R/W	683
SATA0_ERRMR	Error Mask Register **Reset Value:** 0x0000_0000	0x80	R/W	683
SATA Host-Specific: PHY				
SATA0_LLCR	Link Layer Control Register **Reset Value:** 0x0000_0007	0x84	R/W	684
SATA Host-Specific: BIST				
SATA0_RXBISTPD	Received BIST Pattern Definition Register **Reset Value:** 0x0000_0000	0x90	R	685
SATA0_RXBISTD1	Received BIST Data DWORD 1 Register **Reset Value:** 0x0000_0000	0x94	R	685
SATA0_RXBISTD2	Received BIST Data DWORD 2 Register **Reset Value:** 0x0000_0000	0x98	R	686
SATA0_TXBISTPD	Transmit BIST Pattern Definition Register **Reset Value:** 0x0000_0000	0x9C	R/W	686
SATA0_TXBISTD1	Transmit BIST Data DWORD 1 Register **Reset Value:** 0x0000_0000	0xA0	R/W	687
SATA0_TXBISTD2	Transmit BIST Data DWORD 2 Register **Reset Value:** 0x0000_0000	0xA4	R/W	687
SATA0_BISTCR	BIST Control Register **Reset Value:** 0x0000_0000	0xA8	R/W	688
SATA0_BISTFCTR	BIST FIS Count Register **Reset Value:** 0x0000_0000	0xAC	R	688
SATA0_BISTSR	BIST Status Register **Reset Value:** 0x0000_0000	0xB0	R	689
SATA0_BISTDECR	BIST DWORD Error Count Register **Reset Value:** 0x0000_0000	0xB4	R	689
SATA Host-Specific: Miscellaneous				

Table 21-1. SATA Host Controller Register Summary (Continued)

Mnemonic	Register	Address Offset	Access	Page
See **Description** column	Reserved locations **Reset Value:** 0x0000_0000 **Dependencies:** Reads to these locations return zeros; writes have no effect	0xB8–0xF0	–	–
SATA0_TESTR	Test Register **Reset Value:** 0x0000_0000	0xF4	R/W	689
SATA0_VERSIONR	SATA Host Version Register **Reset Value:** HSATA_VERSION_NUM	0xF8	R	690
SATA0_IDR	SATA Host ID Register **Reset Value:** 0x0	0xFC	R	691
–	Unimplemented locations. Access to these locations is illegal and results in bus ERROR response.	0x100–0x3FF	–	–
SATA0_DMADR	FIFO locations in DMA mode. **Reset Value:** none. **Dependencies:** Read only for DMA read/receive operation. Write-only for DMA write/transmit operation.	0x400–0x7FC	R/W	691

21.12 SATA Register Descriptions

Shadow ATA/ATAPI registers (locations SATA0_CDR1 to SATA0_CDR7, SATA0_CLR0) support 8-, 16-, or 32-bit transfer sizes with 32-bit aligned addresses. Non-aligned address accesses to these locations is illegal and return ERROR bus response. Access to these registers should be done according to the ATA/ATAPI specification protocol. For example, writing to the SATA0_CDR1-SATA0_CDR6 registers is prohibited when either the BSY or DRQ bit is set in the Status (SATA0_CDR7) register. Any write to the Command (SATA0_CDR7) register is ignored (bus access is acknowledged normally, but no action is taken) in this case except Device Reset command.

SATA0_CDR1–SATA0_CDR5 registers are used for 48-bit addressing and are implemented as two-byte FIFOs. For example, first write to SATA0_CDR1 results in the Feature register being written, second write to SATA0_CDR1 results in the Feature register being written with a new value, while its previous value is transferred to the Feature expanded register.

During 16- or 32-bit read accesses to SATA0_CDR2–SATA0_CDR5 locations, bits 16:23 return corresponding register value depending on the Device Control register HOB bit (see register description). Bits 0:15 are treated as reserved. Write accesses have no effect, read accesses return zeros.

Other register locations—SATA and SATA Host-specific registers (except SATA0_DBTSR) support 8-, 16-, or 32-bit transfers with addresses naturally-aligned according to the transfer size. For example, SATA0_SCR1 SError register can be accessed as follows (assuming a big endian byte order):

- 32-bit access:
  - SError[0:31]: address SATA0_SCR1
- 16-bit access:
  - Serror[16:31]: address SATA0_SCR1 + 2
  - SError[0:15]: address SATA0_SCR1

- 8-bit access:
  - SError[24:31]: address SATA0_SCR1 + 3
  - SError[16:23]: address SATA0_SCR1 + 2
  - SError[8:15]: address SATA0_SCR1 + 1
  - SError[0:7]: address SATA0_SCR1

The SATA0_DBTSR register supports only 16- or 32-bit transfer sizes for write accesses; 8-bit writes are ignored.

For any access, the transfer size can not exceed 32 bits, otherwise it is considered illegal and returns an ERROR response.

Only the DMADR location in DMA mode supports burst transfers, all other locations should be accessed using single transfers

21.12.1 Data Register (SATA0_CDR0)

This register is used to transfer data from host-to-device and from device-to-host in PIO.

Figure 21-4. Data Register (SATA0_CDR0)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:31	CDR0	This register can only be accessed by the host software when the SATA Host is in the corresponding PIO mode as follows: Read, when the PIO Setup FIS with D = 1 is followed by the Data FIS Write, when the PIO Setup FIS with D = 0 is received.	During PIO read, the software performs a series of reads from this location; during PIO write, the software performs a series of writes to this location. Only single 16-bit bus transfers are supported in this mode.

21.12.2 Error, Feature, Feature Expanded Register (SATA0_CDR1)

This location is used as one of the following:
- Error register when read
- Feature/Feature Expanded register when written. Implemented as 2-byte FIFO.

Figure 21-5. Error, Feature, Feature Expanded Register (SATA0_CDR1)

Bit	Mnemonic	Description	Comments
0:23		Reserved	
24:31	ERROR	Error register containing error/diagnostic information from the device.	
24:31	FEATURE	Current value of the Feature register. Determines the specific function of the SET FEATURES commands.	
24:31	FEATEXP	Previous value of the Feature register (used for 48-bit addressing).	The value is pushed from the Feature register every time SATA0_CDR1 is written.

21.12.3 Sector Count/Sector Count Expanded Register (SATA0_CDR2)

These two 8-bit locations are implemented as a two-byte FIFO and contain the number of sectors for ATA/ATAPI data commands or command-specific parameters on some non-data commands.

Figure 21-6. Sector Count/Sector Count Expanded Register (SATA0_CDR2)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:23	SECCNT	Current value of SATA0_CDR2 when read.	Can be read when Device Control register HOB bit is cleared (SATA0_CLR0[HOB] = 0).
16:23	SECCNTEX	Previous value of SATA0_CDR2 when read.	Can be read when Device Control register HOB bit is set (SATA0_CLR0[HOB] = 1).
24:31	SECCNT	Current value of SATA0_CDR2 register when written.	Device Control register HOB bit is set (SATA0_CLR0[HOB] = 1)
24:31	SECCNTEX	Previous value of SATA0_CDR2 register (used for 48-bit addressing) pushed from the SECCNT register every time SATA0_CDR2 is written.	Device Control register HOB bit is cleared (SATA0_CLR0[HOB] = 0).

21.12.4 Sector Number/Sector Number Expanded Register (SATA0_CDR3)

These two 8-bit locations are implemented as a 2-byte FIFO and contain the start sector number (CHS mode) or LBA low value (LBA mode bits 7:0, 31:24).

Figure 21-7. Sector Number/Sector Number Expanded Register (SATA0_CDR3)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:23	SECNUM	Current value of SATA0_CDR3 when read.	Device Control register HOB bit is set (SATA0_CLR0[HOB] = 1).
16:23	SECNUMEX	Previous value of SATA0_CDR3 register when read.	Device Control register HOB bit is cleared (SATA0_CLR0[HOB] = 0).
24:31	SECNUM	Current value of the SATA0_CDR3 when written. Contains LBA 7:0 bits.	Can be read when SATA0_CLR0[HOB] = 0.
24:31	SECNUMEX	Previous value of the SATA0_CDR3 pushed from the SECNUM every time SATA0_CDR3 is written. Contains LBA 31:24 bits.	Used for 48-bit addressing. Can be read when SATA0_CLR0[HOB] = 1.

21.12.5 Cylinder Low/Cylinder Low Expanded Register (SATA0_CDR4)

These two 8-bit registers are implemented as a two-byte FIFO and contain cylinder number low byte (CHS mode) or LBA mid value (LBA mode bits 15:8, 39:32).

Figure 21-8. Cylinder Low/Cylinder Low Expanded Register (SATA0_CDR4)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:23	CYLLOW	Current value of the SATA0_CDR4 when read.	Device Control register HOB bit is set (SATA0_CLR0[HOB] = 1).
16:23	CYLLOWEX	Previous value of the SATA0_CDR4 register when read.	Device Control register HOB bit is cleared (SATA0_CLR0[HOB] = 0).
24:31	CYLLOW	Current value of the SATA0_CDR4 when written. Contains LBA 15:8 bits.	Can be read when SATA0_CLR0[HOB] = 0.
24:31	CYLLOWEX	Previous value of the SATA0_CDR4 pushed from the CYLLOW every time SATA0_CDR4 is written. Contains LBA 39:32 bits.	Used for 48-bit addressing. Can be read when SATA0_CLR0[HOB] = 1.

21.12.6 Cylinder High/Cylinder High Expanded Register (SATA0_CDR5)

These two 8-bit registers are implemented as a two-byte FIFO and contain cylinder number high byte (CHS mode) or the LBA high value (LBA mode bits 23:16, 47:40).

Figure 21-9. Cylinder High/Cylinder High Expanded Register (SATA0_CDR5)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:23	CYLHIGH	Current value of the SATA0_CDR5 when read.	Device Control register HOB bit is set (SATA0_CLR0[HOB] = 1).
16:23	CYLHIGHEX	Previous value of the SATA0_CDR5 register when read.	Device Control register HOB bit is cleared (SATA0_CLR0[HOB] = 0).
24:31	CYLHIGH	Current value of the SATA0_CDR5 when written. Contains LBA 23:16 bits.	Can be read when SATA0_CLR0[HOB] = 0.
24:31	CYLHIGHEX	Previous value of the SATA0_CDR5 pushed from the CYLLOW every time SATA0_CDR5 is written. Contains LBA 47:40 bits.	Used for 48-bit addressing. Can be read when SATA0_CLR0[HOB] = 1.

21.12.7 Device/Head Register (SATA0_CDR6)

These two 8-bit registers are implemented as a two-byte FIFO and contain cylinder number high byte (CHS mode) or the LBA high value (LBA mode bits 23:16, 47:40).

Figure 21-10. Device/Head Register (SATA0_CDR6)

Bit	Mnemonic	Description	Comments
0:23		Reserved	
24	OBS	Obsolete bit	
25	LBA	Logical Block Addressing: 0 CHS Mode 1 LBA Mode	
26	OBS	Obsolete bit	
27	DEV	Device select bit with the following values: 0 Device 0 (Master) 1 Device 1 (Slave)	This bit should always be cleared (Dev = 0) since the SATA Host implements master-only emulation. If this bit is set, any write to the Command register is ignored, except for a EXECUTE DEVICE DIAGNOSTIC command. A read from the Status/Alternative Status register returns 0x00. (See the ATA/ATAPI specification.) **Note:** This bit is not updated when Register FIS is received from the device. No device can change the state of this bit. This bit is cleared by one of the following conditions: • Power-up • SControl31 = 1 (COMRESET) • Device signals COMINIT • SRST bit set in the Device Control register • EXECUTE DEVICE DIAGNOSTIC command written to the Command register
28:31	HEAD	Head number or other command-dependent information.	

21.12.8 Command/Status Register (SATA0_CDR7)

This location is used as one of the following:
- Command register when written
- Status register when read

Status is a read only 8-bit register and can be written only by the device with either the Register or the Set Device Bits FIS. Read access to the Status register clears the IPF, which negates the ATA interrupt request signal (intrq) to the system bus. Reset occurs on power-on

Figure 21-11. Command/Status Register (SATA0_CDR7)

Bit	Mnemonic	Description	Comments
Command			
0:23		Reserved	
24:31	COMMAND	Command code for device to execute.	A write to this register sets the BSY bit in the Status register (BSY = 1). This register is written last to initiate command execution. Command Register FIS is sent to the device every time this register is written and if both BSY and DRQ bits of the Status register are cleared.
Status			
0:23		Reserved	
24	BSY	Busy bit Device is busy when set.	When BSY and DRQ bits are cleared by the device upon reception of the Register FIS, the host software can access any of the Command Block registers. This bit is cleared on power-on or a COMINIT condition from the device (resulting in Status = 0x7F) and set in the following cases: • Device presence is detected via PHY READY signal assertion (this condition results in Status = 0x80) • SControl31 = 1 (COMRESET condition) • A new command is written to the Command register when BSY = 0 and DEV = 0, or EXECUTE DEVICE DIAGNOSTIC command is written when BSY = 0 and DEV = 1 • DEVICE RESET command is written to the Command register • SRST bit is set in the Device Control register
25	DRDY	Device Ready (device-specific)	
26	DWF	Drive Write Fault (device-specific)	
27	SERV	Service (device specific)	Used in overlap and queued commands.
28	DRQ	Data Request	Asserted when device is ready to transfer data.
29	CORR	Obsolete bit	
30	IDX	Obsolete bit	
31	ERR	Error when high (Error register contains further information	

21.12.9 Alternative Status/Device Control Register (SATA0_CLR0)

This location is used as one of the following:

- Device Control register when written
- Alternative Status register when read. Contains the same value as the Status register (SATA0_CDR7), except the IPF is not cleared when the SATA0_CLR0 is read.

Figure 21-12. Alternative Status/Device Control Register (SATA0_CLR0)

Bit	Mnemonic	Description	Comments
0:23		Reserved	
24	HOB	Read expanded registers (SATA0_CDR2–SATA0_CDR5) when set 0 SATA0_CDR2 SECCNT register is read 1 SATA0_CDR2 SECCNTEX register is read	The HOB bit is cleared after the write access to any Command Block register (SATA0_CDR1–SATA0_CDR7.
25:28		Reserved	
29	SRST	Soft reset bit 0 Device is not reset 1 Device is reset	Control register FIS is transmitted to the device every time the state of this bit is changed.
30	IEN	Interrupt enable bit 0 Enable (intrq is asserted if IPF = 1) 1 Disable (intrq is negated)	The IEN bit is always cleared in all the Host-to-Device Register FISs for better compatibility.
31		Reserved	

21.12.10 SStatus Register (SATA0_SCR0)

This register contains the current state of the interface and host adapter. It is updated continuously and asynchronously by the host adapter. Writes to this register result in bus error response. Reset occurs at power-on.

Figure 21-13. SStatus Register (SATA0_SCR0)

Bit	Mnemonic	Description	Comments
0:19		Reserved	
20:23	IPM	This value indicates the current interface owner management state: 0000 Device not present or communication not established 0001 Interface in active state 0010 Interface in PARTIAL power management state 0110 Interface in SLUMBER power management state All other values reserved.	
24:27	SPD	This value indicates the negotiated interface communication speed established: 0000 No negotiated speed (device not present or communication not established) 0001 Generation 1 communication rate 0010 Generation 2 communication rate All other values reserved.	
28:31	DET	This value indicates the interface device detection and PHY state: 0000 No device detected and PHY communication is not established 0001 Device presence detected but PHY communication not established (PHY COMWAKE signal is detected). 0011 Device presence detected and PHY communication established (PHY READY signal is detected). 0100 PHY in offline mode as a result of the interface being disabled or running in a BIST loopback mode All other values reserved.	

21.12.11 SError Register (SATA0_SCR1)

This register contains supplemental SATA interface error information to complement the error information available in the Shadow Error register. The register represents all the detected errors accumulated since the last time the SError register was cleared. The set bits in the SError register indicate that the corresponding error condition became true one or more times since the last time this bit was cleared. The set bits in the SError register are explicitly cleared by a write operation to the register, or a reset operation (power-on or COMRESET). The value written to clear the set error bits should have ones encoded in the bit positions corresponding to the bits that are to be cleared.

Figure 21-14. SError Register (SATA0_SCR1)

Bit	Mnemonic	Description	Comments
0:3		Reserved	
4	DIAGA	Port Selector Presence detected.	Port selector detection is not supported by SATA Host and this bit always reads zero.
5	DIAGX	Exchanged error Set when PHY COMINIT signal is detected.	
6	DIAGF	Unrecognized FIS type Set when the Transport Layer receives a FIS with good CRC, but unrecognized FIS type.	
7	DIAGS	Transport state transition error Set when the Transport Layer detects one of the following conditions: • Wrong sequence of received FISs • PIO count mismatch between the PIO Setup FIS and the following Data FIS • Odd PIO/DMA byte count or DMA buffer offset • Wrong FIS length	
8	DIAGT	Link sequence (illegal transition) error Set when the Link Layer detects an erroneous Link state machine transition.	
9	DIAGH	Handshake error Set when the Link Layer receives one or more R_ERRp handshake responses from the device after frame transmission.	
10	DIAGC	CRC error Set when the Link Layer detects CRC error in the received frame.	
11	DIAGT	Disparity error Set when the Link Layer detects incorrect disparity in the received data.	
12	DIAGB	10b to 8b decoder error Set when the Link Layer detects 10b to 8b decoder error in the received data.	
13	DIAGW	Comm Wake Set when Phy COMWAKE signal is detected.	

Figure 21-14. SError Register (SATA0_SCR1)

Bit	Mnemonic	Description	Comments
14	DIAGI	PHY internal error Set when the PHY detects an internal error, as indicated by the assertion of the phy_rx_err input.	Setting this bit is controlled by the LLCR.ERREN bit: • If ERREN = 0 (default), only errors occurring inside the received FIS set the DIAG_I bit. • If ERREN = 1, any error inside or outside the FIS sets the DIAG_I bit.
15	DIAGN	PHYRdy change Set when Phy READY signal state is changed.	
16:19		Reserved	
20	ERRE	Internal host adapter error Set when any one of the illegal accesses is attempted on the AHB bus (see list of errors in **Comments** column at right), or dma_finish_tx is asserted, but no data had been written during DMA write operation.	Address is not 32-bit aligned when accessing SATA0_CDR0–SATA0_CDR7, SATA0_CLR0, SATA0_DMADR locations. Write access to the read only locations (for example, SStatus, SATA0_FPTR, SATA0_FPBOR, SATA0_FPTCR.) Address selects non-implemented area of the address space. Shadow ATA registers write access when either BSY or DRQ bit is set in Status register. Transfer size exceeds 32 bits. Access to the FIFO (SATA0_CDR0/SATA0_DMADR) and SATA Host not in PIO or DMA mode. Burst access to the FIFO (SATA0_CDR0) in PIO mode or to registers if the burst size exceeds one (burst size of one is acceptable). Access to the FIFO (SATA0_CDR0/SATA0_DMADR) in PIO or DMA mode when the RxFIFO is empty, or the TxFIFO is full. This error might result from the AHB burst size exceeding the value programmed in the SATA0_DBTSR register that sets the Rx/TxFIFO threshold values for DMA flow control. DMA bus transfer starts only when there is enough data in the RxFIFO or enough space in the TxFIFO for the requested DMA transaction. In PIO mode, the error is generated if the device supplies read data (or accepts write data) at a rate slower than 2.3 MB/s (hclk = 300MHz). DMA transaction transfer size is 16 bits.
21	ERRP	Protocol error Set when any of the following error conditions is detected: • Transport state transition error (DIAGT) • Unrecognized FIS type (DIAGF) • Link sequence error (DIAGS) • RxFIFO overflow condition	
22	ERRC	Non-recovered persistent communication or data integrity error Set when PHY READY signal is negated (PHY Not Ready condition) due to the loss of communication with the device or problems with interface, but not after the transition from active to PARTIAL or SLUMBER power management state.	

Figure 21-14. SError Register (SATA0_SCR1)

Bit	Mnemonic	Description	Comments
23	ERRT	Non-recovered transient data integrity error Set if any of the 10b to 8b decoder error (DIAGB), CRC error (DIAGC), Disparity error (DIAGD), Handshake error (DIAGH), Transport transition error (DIAGT), or RxFIFO overflow is detected in the Data FIS, or non-data FIS transmission was not recovered after three retries.	
24:29		Reserved	
30	ERRM	Recovered communication error Set when PHY READY condition is detected after interface initialization, but not after transition from PARTIAL or SLUMBER power management state to active state.	
31	ERRI	Recovered data integrity error Set when non-data FIS transmission is unsuccessful (such as when a DIAGH error is detected), but is recovered after one to three retries.	

21.12.12 SControl Register (SATA0_SCR2)

This register provides control for the SATA interface. Write operations to the SControl register result in an action being taken by the host adapter or interface. Read operations from the register return the last value written to it.

Note: These bits are static and should not be changed frequently due to the clock-crossing from the Transport-to-Link Layers. Software must wait for at least seven periods of the slower clock (clk_asic or hclk) before changing this register.

Figure 21-15. SControl Register (SATA0_SCR2)			
Bit	**Mnemonic**	**Description**	**Comments**
0:11		Reserved	
12:15	PMP	The Port Multiplier Port (PMP) field represents the 4-bit value that is placed in the PM Port field of all transmitted FISs.	
16:19	SPM	The Select Power management field (SPM) is used to select a power management state. A non-zero value written to this field causes the power management state specified to be initiated. A value written to this field is treated as a one-shot. It is read as 0b0000. 0000 No power management state transition requested. 0001 Transition to PARTIAL power management state initiated. 0010 Transition to SLUMBER power management state initiated. 0100 Transition to the active power management state initiated. All other values reserved.	Normally system software should wait until the interface is in the active state using SStatus register (SATA0_SCR0[DET] = 0b0011 or SATA0_SCR0[IPM] = 0b0001) before initiating either PARTIAL or SLUMBER power management state. Otherwise, the corresponding signal (phy_partial or phy_slumber) is asserted without PMREQ/PMACK handshaking protocol. This feature can be used to put the PHY into power management state even if a device is not connected.
20:23	IPM	This field represents the enabled interface power management states that can be invoked with the SATA interface power management capabilities: 0000 No interface power management state restrictions 0001 Transitions to the PARTIAL state disabled 0010 Transitions to the SLUMBER state disabled 0011 Transitions to both the PARTIAL and SLUMBER states disabled All other values reserved.	
24:27	SPD	This field represents the highest-allowed communication speed that the interface is allowed to negotiate when the speed is established: 0000 No speed negotiation restrictions 0001 Limit speed negotiation to a rate not greater than Generation 1 communication rate 0010 Limit speed negotiation to a rate not greater than Generation 2 communication rate All other values reserved.	If the host software needs to change this field's value, it should reset the interface (SATA0_SCR2.DET = 0b0001) at the same time to ensure proper speed negotiation.
28:31	DET	This field controls the host adapter device detection and interface initialization: 0000 No device detection or initialization action requested 0001 Perform interface communication initialization sequence to establish communication. This is functionally equivalent to a hard reset and results in the interface being reset and communication reinitialized (COMRESET condition). 0100 Disable SATA interface and put PHY in offline mode. All other values reserved.	

21.12.13 SActive Register (SATA0_SCR3)

This register is used for native SATA command queuing and contains the information returned in the SActive field of the Set Device Bits FIS.

The host software can set bits in the SActive register by a write operation to this register. The value written to the set bits should have ones encoded in the bit positions corresponding to the bits that are to be set. Bits in the SActive register can not be cleared as a result of a register write operation by the host, and host software cannot clear bits in the SActive register.

Set bits in the SActive register are cleared as a result of data returned by the device in the SActive field of the Set Device Bits FIS. The value returned in this field has ones encoded in the bit positions corresponding to the bits that are to be cleared. The device cannot set bits in this register. All bits in the SActive register are cleared upon issuing a hard reset (COMRESET) signal or as a result of issuing a software reset by setting SRST bit of the Device Control register.

For the native command queuing protocol, the SActive value represents the set of outstanding queued commands that have not completed successfully yet. The value is bit-significant and each bit position represents the status of a pending queued command with a corresponding TAG value.

Figure 21-16. SActive Register (SATA0_SCR3)

Bit	Mnemonic	Description	Comments
0:31		SActive field of the Set Device Bits FIS	

21.12.14 SNotification Register (SATA0_SCR4)

This register is used to notify the host software which devices have sent a Set Device Bits FIS with Notification bit set. When a Set Device Bits FIS with Notification bit set to 1 is received, the bit corresponding to the value of the PM Port field in the FIS is set, and an interrupt is generated if the I bit in the FIS is set and interrupt is enabled (nIEN = 0 in the Device Control register).

Set bits in the SNotification register are explicitly cleared by a write operation to the SNotification register, or a power-on reset. The register is not cleared due to a COMRESET condition. The value written to clear set bits should have ones encoded in the bit positions corresponding to the bits that are to be cleared.

Figure 21-17. SNotification Register (SATA0_SCR4)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:31	NOTIFY	Represents whether a particular device with the corresponding PM Port number has sent a Set Device Bits FIS to the host with the Notification bit set.	

21.12.15 Not applicable (Reserved for SATA) (SATA0_SCR5–SATA0_SRC15)

Figure 21-18. Not applicable (Reserved for SATA) (SATA0_SCR5–SATA0_SRC15)

Bit	Mnemonic	Description	Comments
0:31		Reserved	

21.12.16 First-Party DMA Tag Register (SATA0_FPTAGR)

This register contains the command 5-bit TAG value, which is updated every time a new DMA Setup FIS is received. The DMA controller uses this value to identify the buffer region in the host system memory selected for the data transfer. Write access to this location results in the bus error response.

Figure 21-19. First-Party DMA Tag Register (SATA0_FPTAGR)

Bit	Mnemonic	Description	Comments
0:26		Reserved	
27:31	TAG	First-party DMA TAG value Updated every time a new DMA Setup FIS is received from the device.	

21.12.17 First-Party DMA Buffer Offset Register (SATA0_FPBOR)

This register contains the DMA buffer offset value, which is updated every time a new DMA Setup FIS is received. The device uses the offset to transfer DMA data out of order. Bits 1 and 0 should always be cleared (32-bit-aligned offset). A write access to this location results in the bus error response.

Figure 21-20. First-Party DMA Buffer Offset Register (SATA0_FPBOR)

Bit	Mnemonic	Description	Comments
0:31	TAG	Buffer Offset	

21.12.18 First-Party DMA Transfer Count Register (SATA0_FPTCR)

This register contains the number of bytes that is transferred. It is updated every time a new DMA Setup FIS is received. Bit 0 should always be cleared (even number of bytes and the value should be non-zero). A write access to this location results in the bus error response.

Figure 21-21. First-Party DMA Transfer Count Register (SATA0_FPTCR)

Bit	Mnemonic	Description	Comments
0:31	TAG	Buffer Offset	

21.12.19 DMA Control Register (SATA0_DMACR)

This register contains bits indicating the status of the DMA transmit or receive channel. Power-on or COMRESET clears this register.

Note: Both TXCHEN and RXCHEN bits are cleared by the SATA Host after the First-party DMA command data phase ends (all the data for this command has been transferred).

Figure 21-22. DMA Control Register (SATA0_DMACR)

Bit	Mnemonic	Description	Comments
0:28		Reserved	
29	TXMODE	Transmit Mode 0 Tx Data FIS is closed when dma_finish_tx is asserted at the end of the DMA block transfer. 1 Tx Data FIS is closed when TXCHEN bit is cleared at the end of the DMA command transfer. Dma_finish_tx input is ignored in this case.	
30	RXCHEN	Receive Channel Enable 0 DMA receive channel is disabled 1 DMA receive channel is enabled and ready for transfer	Software must set this bit prior to issuing a DMA read command to the device and clear this bit after DMA controller has finished transferring data for this command. **Note:** If this bit is cleared during data transfer and SError bits ERRP, DIAGC, DIAGB, and DIAGD are all cleared, the Link SYNC-escapes the current FIS. If any of these SError bits is set when RXCHEN is cleared, the current Data FIS is flushed from the RxFIFO until the TSM detects the End Status DWORD. This feature is useful for software error recovery operation.
31	TXCHEN	Transmit Channel Enable 0 DMA transmit channel is disabled 1 DMA transmit channel is enabled and ready for transfer	Software must set this bit prior to issuing a DMA write command to the device and clear this bit after DMA controller has finished transferring data for this command.

21.12.20 DMA Burst Transaction Size Register (SATA0_DBTSR)

This register is used to set RxFIFO "pop almost empty" and TxFIFO "push almost full" thresholds to the burst transaction size (in DWORDs) for a DMA read or write operation. The SATA Host generates corresponding request signals (dma_req_rx or dma_req_tx) to the DMA controller as follows:

- dma_req_rx is asserted when RxFIFO contains enough data for the burst transaction of MRD size
- dma_req_tx is asserted when TxFIFO contains enough free space for the burst transaction of MWR size

Power-up or COMRESET condition initializes this register to the value shown above.

The MRD/ MWR field can only be written if the corresponding RXCHEN/ TXCHEN bit of the SATA0_DMACR register is cleared.

Note: It is important that the DMA Read/Write burst transaction size never exceeds MRD/MWR values. Otherwise, an ERROR response is generated by the slave interface if either the RxFIFO-empty or the TxFIFO-full condition is detected during DMA bus transfer. Host software must ensure that the DMA controller is programmed with the same values prior to enabling a channel for transfer.

This location supports only 16- or 32-bit transfer sizes for write accesses. Eight-bit write accesses are ignored. MRD and MWR fields can be accessed separately using 16-bit wide transfers with naturally-aligned addresses:

- DBTSR for MWR field
- DBTSR+2 for MRD field

Figure 21-23. DMA Burst Transaction Size Register (SATA0_DBTSR)

Bit	Mnemonic	Description	Comments
0:8		Reserved	
9:15		This field is used to set the RxFIFO "pop almost empty" flag to the maximum burst size in DWORDs for the DMA read operation. Can be written if SATA0_DMACR[RXCHEN] = 0, otherwise, the write to this field is ignored. **Note:** MRD = 0 might result in a bus error response during DMA read transaction. Upper boundary is derived from the fact that the device stops sending data when the host generates HOLDp raf DWORDs from the RxFIFO "full" condition. This might result in possible lock condition (dma_req_rx is never generated) if this value is exceeded. **m** represents Rx FIFO RAM address width as follows: **m** = Log2(RXFIFO_DEPTH), where RXFIFO_DEPTH is 128 **raf** = 43 Defaults to 20 DWORDs on reset (64 – 43 - 1 = 20) **Range:** 1 to (RXFIFO_DEPTH – raf – 1 = 84 or 0x54)	
16:24		Reserved	
25:31		This field is used to set the TxFIFO "push almost full" flag to the maximum burst size in DWORDs for the DMA write operation. Can be written if SATA0_DMACR[TXCHEN] = 0. Otherwise, the write to this field is ignored. **Note:** MWR = 0 might result in bus error response during DMA write transaction. **k** represents Tx FIFO RAM address width as follows: **k** = Log2(TXFIFO_DEPTH), where TXFIFO_DEPTH is 128 Defaults to 16 DWORDs on reset (32/2 = 16) **Range:** 1 to (TXFIFO_DEPTH –2 = 126 or 0x7E)	

21.12.21 Interrupt Pending Register (SATA0_INTPR)

This register contains all SATA Host interrupt events before masking. The bits are set by an interrupt event. All the interrupt bits together with the IPF ATA interrupt flag are ORed to generate the SATA Host intrq output. The set bits in the this register can be cleared by a write operation to the register, or a reset operation (power-on or COMRESET). The value written to clear set bits should have ones encoded in the bit positions corresponding to the bits that are to be cleared.

Figure 21-24. Interrupt Pending Register (SATA0_INTPR)

Bit	Mnemonic	Description	Comments
0	IPF	This bit reflects the state of the ATA Interrupt Pending Flag (IPF). It is set when an FIS with I = 1 is received from the Device, and cleared when Status register SATA0_CDR7 is read.	
1:4		Reserved	
5:15	ERRADDR	This field contains lower 11 bits of haddr bus (bits [21:31]) of an illegal bus access detected by the SATA Host AHB slave interface, or 0x3FF value if dma_finish_tx is asserted, but no data had been written to the DMADR location during DMA write operation.	These bits can not be cleared by a write operation.
16:23		Reserved	
24	CMDGOOD	Set when a good Register or Set Device Bits FIS with Status.ERR bit cleared (bit 16 of the first DWORD) is received from the Device.	
25	CMDABORT	Set when a good Register or Set Device Bits FIS with Status.ERR bit set (bit 16 of the first DWORD) is received from the Device.	
26	PRIMERR	Set when the Link Layer detects an invalid K-character (not K28.5 or K28.3) in a primitive.	
27	NEWBIST	Set when a supported BIST Activate FIS is received from the device without errors.	
28	ERR	Set when any of the bits in the SError register is set and the corresponding bit in the SATA0_ERRMR register is set.	This bit can not be cleared by a write operation since it reflects the state of the SError register. It is cleared if either SError register bits are cleared or masked with the SATA0_ERRMR register.
29	PMABORT	Set when the Link Layer detects a power mode abort condition (power mode is aborted by the device requesting a frame transmission).	Must be cleared explicitly by software before issuing a power management request to the interface.
30	NEWFP	Set when a DMA Setup FIS is received from the device without errors.	
31	DMAT	Set when DMATp is received from the device during Data FIS transmission.	

21.12.22 Interrupt Mask Register (SATA0_INTMR)

This register is used to mask or enable corresponding interrupt events in the INTPR register. An interrupt event is masked if the bit is cleared and is enabled if set. If any of the INTPR bits are set or if IPF is set and enabled (unmasked), the intrq output is asserted. A COMRESET condition clears this register.

Figure 21-25. Interrupt Mask Register (SATA0_INTMR)

Bit	Mnemonic	Description	Comments
0:23		Reserved	
24	CMDGOODM	CMDGOOD interrupt 1 Enabled 0 Masked	
25	CMDABORTM	CMDABORT interrupt 1 Enabled 0 Masked	
26	PRIMERRM	PRIMERR interrupt 1 Enabled 0 Masked	
27	NEWBISTM	NEWBIST interrupt 1 Enabled 0 Masked	
28	ERRM	ERR interrupt 1 Enabled 0 Masked	
29	PMABORTM	PMABORT interrupt 1 Enabled 0 Masked	
30	NEWFPM	NEWFP interrupt 1 Enabled 0 Masked	
31	DMATM	DMAT interrupt 1 Enabled 0 Masked	

21.12.23 Error Mask Register (SATA0_ERRMR)

This register is used to mask or enable corresponding bits of the SError register prior to setting the ERR bit of the SATA0_INTPR. This allows driver software to select the SError bits that can cause the interrupt output intrq to be asserted. The SATA0_INTPR ERR bit is set if any of the SError bits are set and the corresponding SATA0_ERRMR bit is set. Clearing the SATA0_ERRMR bit would mask the corresponding SError bit from setting the SATA0_INTRP ERR bit. COMRESET condition clears this register.

Figure 21-26. Error Mask Register (SATA0_ERRMR)

Bit	Mnemonic	Description	Comments
0:31		Corresponding SError bit	

21.12.24 Link Layer Control Register (SATA0_LLCR)

This register provides Link Layer (LL) control capability for the host software. Power-on or COMRESET condition sets SCRAM, DESCRAM, and RPD bits. The SCRAM bit is cleared when the SATA Host enters a BIST device far-end transmit mode with scrambling bypassed.

Note: These bits are static and should not be changed frequently due to the clock-crossing from Transport to Link Layers. Software must wait at least seven periods of the slower clock (clk_asic or hclk) before changing this register.

Figure 21-27. Link Layer Control Register (SATA0_LLCR)

Bit	Mnemonic	Description	Comments
0:26		Reserved	
27	ERREN	Error Enable RW This bit allows or filters (disables) PHY Internal errors outside the FIS boundary to set corresponding SError bits. 0 Filter errors outside the FIS, allow errors inside the FIS 1 Allow errors outside or inside the FIS.	
28		Reserved	
29	RPD	Repeat primitive drop function 1 Enabled 0 Disabled	
30	DESCRAM	Descrambler 1 Enabled 0 Disabled	
31	SCRAM	Scrambler 1 Enabled 0 Disabled	

21.12.25 Received BIST Pattern Definition Register (SATA0_RXBISTPD)

This register contains the pattern definition field of the received BIST Activate FIS (bits 23:16 of the first DWORD). This field defines the SATA Host loopback mode requested by the device. It is updated every time a new BIST Activate FIS is received from the device. A write access to this location results in a bus error response.

Figure 21-28. Received BIST Pattern Definition Register (SATA0_RXBISTPD)

Bit	Mnemonic	Description	Comments
0:23		Reserved	
24:31	RXPD	Pattern definition field of the received BIST Activate FIS (bits 23:16 of the first DWORD). It is used to put the SATA Host in one of the following BIST modes: 0x10 Far-end retimed 0x08 Far-end analog (if PHY supports this mode) 0x80 Far-end transmit only 0xA0 Far-end transmit only with scrambler bypassed All other values should not be used by the device, otherwise the FIS is negatively acknowledged with R_ERRp.	For far-end transmit only modes, the SATA0_RXBISTD1 register would contain the required data pattern.

21.12.26 Received BIST Data DWORD 1 Register (SATA0_RXBISTD1)

This register contains the second DWORD (first data DWORD) of the received BIST Activate FIS. It is updated every time a new three-DWORD BIST Activate FIS is received from the device and selects a pattern for far-end transmit only loopback mode. A write access to this location results in a bus error response.

The following patterns are supported by the SATA Host (SATA0_RXBISTD1[31:0]):
- 0xEE27FEF1: Low transition density pattern
- 0xB5B5B5B5: High transition density pattern
- 0xABABABAB: Low frequency spectral component pattern
- 0x7F7F7F7F: Simultaneous switching outputs pattern

If none of these values is detected in the SATA0_RXBISTD1 register, simultaneous switching pattern is transmitted by default.

Figure 21-29. Received BIST Data DWORD 1 Register (SATA0_RXBISTD1)

Bit	Mnemonic	Description	Comments
0:31		Data DWORD 1	

21.12.27 Received BIST Data DWORD 2 Register (SATA0_RXBISTD2)

This register contains the third DWORD (second data DWORD) of the received BIST Activate FIS. It is updated every time a new three-DWORD BIST Activate FIS is received from the device. A write access to this location results in a bus error response.

Figure 21-30. Received BIST Data DWORD 2 Register (SATA0_RXBISTD2)

Bit	Mnemonic	Description	Comments
0:31		Data DWORD 2	

21.12.28 Transmit BIST Pattern Definition Register (SATA0_TXBISTPD)

This register contains the pattern definition field in bits 7:0 of the transmit BIST Activate FIS. The host software puts the device in one of the BIST modes by write access to this register.

Note: Host software should reset the interface by toggling SControl31 and wait for the BSY and DRQ bits to be cleared prior to requesting any of the BIST modes. The device must support the BIST mode requested by the host software, otherwise the result is indeterminate.

Figure 21-31. Transmit BIST Pattern Definition Register (SATA0_TXBISTPD)

Bit	Mnemonic	Description	Comments
0:23		Reserved	
24:31	TXPD	Pattern definition field of the transmit BIST Activate FIS (bits 23:16 of the first DWORD). It is used to put the device in one of the loopback modes as defined in SATA specification: Bit 31 (V): Vendor-specific Bit 30 (R): Reserved Bit 29 (P): Primitive bit (valid only with the T-bit) Bit 28 (F): Far-end analog loopback Bit 27 (L): Far-end retimed loopback Bit 26 (S): Bypass scrambling (valid only with T-bit) Bit 25 (A): Bypass align (valid only with T-bit) Bit 24 (T): Far-end transmit only Each write access causes BIST Activate FIS to be transmitted to the device. Bits 28 (F), 27 (L), and 24 (F) are mutually exclusive and should not be set at the same time, otherwise the BIST request is ignored.	For far-end transmit only mode (T = 1) the software has to load SATA0_TXBISTD1 and SATA0_TXBISTD2 registers with the required pattern first, then write to the SATA0_TXBISTPD.

21.12.29 Transmit BIST Data DWORD 1 Register (SATA0_TXBISTD1)

This register contains the first data DWORD with the BIST pattern. It is updated by the host software prior to requesting a new three-DWORD BIST Activate FIS transmission to the device (by writing to SATA0_TXBISTPD with bit 24 set).

Figure 21-32. Transmit BIST Data DWORD 1 Register (SATA0_TXBISTD1)

Bit	Mnemonic	Description	Comments
0:31		Data DWORD 1	

21.12.30 Transmit BIST Data DWORD 2 Register (SATA0_TXBISTD2)

This register contains the second data DWORD with the BIST pattern. It is updated by the host software prior to requesting a new three-DWORD BIST Activate FIS transmission to the device (by writing to SATA0_TXBISTPD with bit 24 set).

Figure 21-33. Transmit BIST Data DWORD 2 Register (SATA0_TXBISTD2)

Bit	Mnemonic	Description	Comments
0:31		Data DWORD 2	

21.12.31 BIST Control Register (SATA0_BISTCR)

This register is used in BIST initiator modes and is loaded by the host software prior to sending BIST Activate FIS to the device (by means of SATA0_TXBISTPD write).

Figure 21-34. BIST Control Register (SATA0_BISTCR)

Bit	Mnemonic	Description	Comments
0:13		Reserved	
14	CNTCLR	Used to clear BIST error count registers and is treated as a one-shot and read as zero. 1 Clear registers: SATA0_BISTFCTR, SATA0_BISTSR, SATA0_BISTDECR.	
15	NEALB	Used to put the host PHY into near-end analog loopback mode and is treated as a one-shot and read as zero. 1 Near-end analog loopback request. PATTERN field contains the appropriate pattern.	This mode should be initiated either in the PARTIAL or SLUMBER power mode, or with the device disconnected from the host PHY (Link NOCOMM state). BIST Activate FIS is not sent to the device in this mode.
16:28		Reserved	
29:31	PATTERN	Defines one of the following SATA compliant patterns to be transmitted by the SATA Host to the device in far-end retimed/ far-end analog modes, or to the host PHY in near-end analog mode. 000 Simultaneous switching outputs bit pattern 001 High transition density bit pattern 010 Low transition bit pattern 011 Low frequency spectral component bit pattern 100 Composite pattern All other values are reserved.	

21.12.32 BIST FIS Count Register (SATA0_BISTFCTR)

This register contains the received BIST FIS count in the loopback initiator far-end retimed, far-end analog and near-end analog modes. It is updated each time a new BIST FIS is received. It is cleared by COMRESET or by software setting the SATA0_BISTCR register bit CNTCLR. This register does not roll over and freezes when the 0xFFFF_FFFF value is reached. It takes approximately 66 hours of continuous BIST operation at Gen. 1 speed to reach this value. Write access to this location results in a bus error response.

Figure 21-35. BIST FIS Count Register (SATA0_BISTFCTR)

Bit	Mnemonic	Description	Comments
0:31		Received BIST FIS count	

21.12.33 BIST Status Register (SATA0_BISTSR)

This register contains errors detected in the received BIST FIS in the loopback initiator far-end retimed, far-end analog and near-end analog modes. It is updated each time a new BIST FIS is received. It is cleared by a COMRESET condition or by the software setting the SATA0_BISTCR register bit CNTCLR.

Figure 21-36. BIST Status Register (SATA0_BISTSR)

Bit	Mnemonic	Description	Comments
0:7		Reserved	
8:15	BRSTERR	Burst error count Accumulated each time a burst error condition is detected: DWORD error is detected in the received frame and 1.5 seconds (27,000 frames) passed since the previous burst error was detected. Value does not roll over and freezes at 0xFF.	
16:31	FRAMERR	Frame error count Accumulated (new value is added to the old value) each time a new BIST frame with a CRC error is received. Value does not roll over and freezes at 0xFFFF.	

21.12.34 BIST DWORD Error Count Register (SATA0_BISTDECR)

This register contains the number of DWORD errors detected in the received BIST frame in the loopback initiator far-end retimed, far-end analog and near-end analog modes. It is updated each time a new BIST frame is received. It is cleared by a COMRESET condition or by the software setting the SATA0_BISTCR register bit CNTCLR. A write access to this location results in bus error response.

Figure 21-37. BIST DWORD Error Count Register (SATA0_BISTDECR)

Bit	Mnemonic	Description	Comments
0:31	DWERR	DWORD error count Accumulated (new value is added to the old value) each time a new BIST frame is received. Value does not roll over and freezes if it exceeds 0xFFFF_F000.	

21.12.35 Test Register (SATA0_TESTR)

This register is used to put the SATA Host slave interface into a test mode; a condition where the read back value of all the registers match the value written. In normal operation the read back value of some registers is a function of the SATA Host state and does not match the value written. This register is not reset by the COMRESET condition.

The registers are treated as follows when TEST_IF = 1:

- SATA0_CDR0 becomes 32-bit Read/Write register. Only single transfers are allowed. Transfer size should be 32 bits.

- SATA0_CDR1 becomes Read/Write register similar to SATA0_CDR2–SATA0_CDR5. SATA0_CLR0 HOB bit state determines which register is read when accessing SATA0_CDR1–SATA0_CDR5: HOB = 1: Feature expanded register, HOB = 0: Feature register and so on. 16-bit read access to SATA0_CDR1–SATA0_CDR5 returns both registers as follows (using SATA0_CDR1 as example):
  - HOB = 1: 15:8 = Feature register, 7:0 = Feature expanded register
  - HOB = 0: 15:8 = Feature expanded register, 7:0 = Feature register
- SATA0_CDR7 and SATA0_CLR0 become Read/Write registers.
- All read only registers (for example, SATA0_SCR0, SATA0_FPTAGR) except SATA0_VERSIONR, SATA0_IDR and the ERR_ADDR field of the SATA0_INTPR become Read/Write registers. Writes to VERSIONR, IDR and the ERR_ADDR field of INTPR registers are ignored.
- SATA0_DMADR locations become illegal and return ERROR response if accessed.

Note: If the test interface mode is entered with the device connected to the host adapter, the interface should be reset by setting SControl31 = 1 (COMRESET) after TEST_IF bit is cleared.

Figure 21-38. Test Register (SATA0_TESTR)

Bit	Mnemonic	Description	Comments
0:14		Reserved	
15	BSYCLR	Used to clear Status BSY and DRQ bits by software when set Treated as one-shot type and read as zero.	This bit should not be used for normal operation. It is intended only for error recovery or debug.
16:29		Reserved	
30	TIEN	Test Interrupt Enable: 0 Intrq output is not asserted in test mode when TEST_IF = 1. 1 Intrq output is asserted in test mode when TEST_IF = 1 and any of the INTPR bits and corresponding INTMR bits are set.	
31	TESTIF	0 Normal mode. 1 Test mode: the read back value of all the registers matches the value written. Normal operation is disabled.	

21.12.36 Version Register (SATA0_VERSIONR)

This 32-bit read only register contains hard-coded value 0x3138 322A. The value represents an ASCII code of the version number. For example, version 1.00* is coded as 0x3130_302A. A write access to this register results in the bus error response.

Figure 21-39. Version Register (SATA0_VERSIONR)

Bit	Mnemonic	Description	Comments
0:31		Version number	

21.12.37 ID Register (SATA0_IDR)

This register contains hard-coded value of 0x0. A write access to this register results in the bus error response.

Figure 21-40. Version Register (SATA0_IDR)

Bit	Mnemonic	Description	Comments
0:31		ID number	

21.12.38 FIFO Location in DMA Mode Register (SATA0_DMADR)

This location is used to transfer data from host to device and from device to host in DMA mode.

Note: Transfer size should be maintained during the whole DMA transfer. Starting address of the burst transfer should always be 0x4 BFFD 1400, except when the burst is interrupted by the system arbiter (early burst termination) and continued with the next address within the 0x4 BFFD 1400 to 0x4 BFFD 17FC range. The DMA address can be fixed to 0x4 BFFD 1400 for all the beats in the burst. The burst addresses must be 32-bit aligned.

Figure 21-41. FIFO Location in DMA Mode Register (SATA0_DMADR)

Bit	Mnemonic	Description	Comments
0:31	DMAADR	DMA FIFO location	This location can only be accessed by the host software or DMA controller when the SATA Host is in the corresponding DMA mode: • Read (when the Data FIS is being received) • Write (when the DMA Activate or DMA Setup FIS is received), and the corresponding DMA handshake signal is asserted (dma_req or dma_single). Both single and burst 32- bit or 16-bit bus transfers are supported in this mode.

21.13 SATA DMA

The SATA function has a DMA capability separate from the DMA provided for other functions in this chip. The SATA DMA can be programmed through software registers as described in *SATA DMA Register Summary* on page 717.

21.13.1 Register Access

All registers in this chip are 32 bits wide. Some of the SATA DMA registers require 64 bits and appear as two 32-bit registers designated as Low and High. A write to any bits labeled as "Reserved" within a register is ignored. A read from Reserved bits in the register reads back 0.

21.13.2 Illegal Register Access

An illegal access can be any of the following (all of the registers identified in the following bullets are AHBDMA0_ registers):

- An AHB transfer of hsize greater than 64 is attempted.

- The address does not decode to a valid address.
- A write to the SAR0, DAR0, LLP0, CTL0, SGR0, or DSR0 registers occurs when the channel is enabled.
- A read from the ClearBlock, ClearDstTran, ClearErr, ClearSrcTran, or ClearTfr register is attempted.
- A write to the StatusBlock, StatusDstTran, StatusErr, StatusSrcTran, or StatusTfr register is attempted.
- A write to the StatusInt register is attempted.
- A write to either the DmaIdReg or DMA Component Type or DMA Component Version registers is attempted.

An illegal access (read/write) returns an error response.

21.13.3 SATA DMA Transfer Types

A DMA transfer may consist of single or multiblock transfers. On successive blocks of a multiblock transfer, the AHBDMA0_SAR0/DAR0 register is reprogrammed using either of the following methods:
- Block chaining using linked lists
- Auto-reloading
- Contiguous address between blocks

On successive blocks of a multiblock transfer, the AHBDMA0_CTL0_L/H register is reprogrammed using either of the following methods:
- Block chaining using linked lists
- Auto-reloading

When block chaining, using Linked Lists is the multiblock method of choice. On successive blocks, the AHBDMA0_LLP0 register is reprogrammed using block chaining with linked lists.

A block descriptor consists of six AHBDMA0_ registers: SAR0, DAR0, LLP0, CTL0_L/H, SSTAT0, and DSTAT0. The first four registers, along with the CFG0_L/H registers, are used to set up and describe the block transfer.

Note: The term Link List Item (LLI) and block descriptor are synonymous.

21.13.3.1 Multiblock Transfers (MBTs)

The SATA DMA reprograms the channel registers prior to the start of each block by fetching the block descriptor for that block from system memory. This is known as an LLI update.

SATA DMA block chaining uses a Linked List Pointer register (AHBDMA0_LLP0) that stores the address in memory of the next linked list item. Each LLI contains the corresponding block descriptors:

1. AHBDMA0_SAR0
2. AHBDMA0_DAR0
3. AHBDMA0_LLP0
4. AHBDMA0_CTL0_L/H
5. AHBDMA0_SSTAT0
6. AHBDMA0_DSTAT0

To set up block chaining, you program a sequence of Linked Lists in memory.

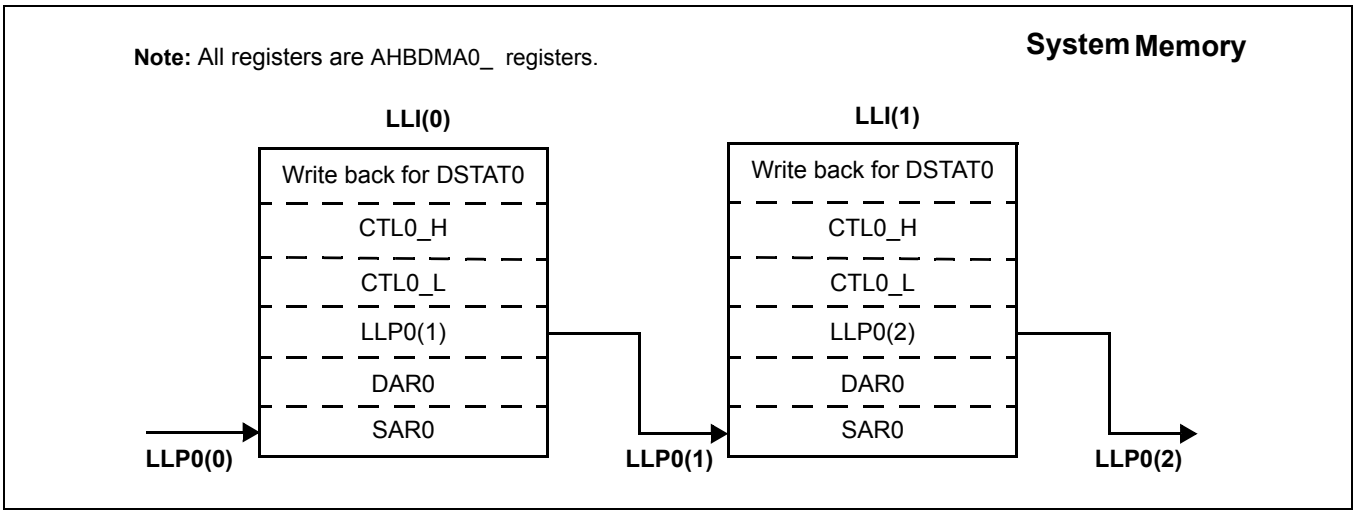
LLI accesses are always 32-bit accesses (Hsize = 2) and cannot be changed or programmed to anything other than 32 bits.

The AHBDMA0_ registers SAR0, DAR0, LLP0, and CTL0_L/H are fetched from system memory on an LLI update. *Figure 21-42* shows how you use chained linked lists in memory to define multiblock transfers using block chaining.

The order of a Linked List item is as follows:

1. AHBDMA0_SAR0
2. AHBDMA0_DAR0
3. AHBDMA0_LLP0
4. AHBDMA0_CTL0_L/H
5. AHBDMA0_DSTAT0

Figure 21-42. Multiblock Transfer Using Linked Lists



Note: In order to not confuse the AHBDMA0_ SAR0, DAR0, LLP0, CTL0_L/H, STAT0, and DSTAT0 register locations of the LLI with the corresponding SATA DMA memory mapped register locations, the LLI register locations are prefixed with LLI; that is, LLI.AHBDMA0_SAR0, LLI.AHBDMA0_DAR0, LLI.AHBDMA0_LLP0, LLI.AHBDMA0_CTL0_L/H, LLI.AHBDMA0_SSTAT0, and LLI.AHBDMA0_DSTAT0.

Figure 21-43 shows the mapping of a Linked List Item stored in memory to the channel registers block descriptor.

Rows 6 through 10 of *Table 21-2* show the required values of AHBDMA0_LLP0, AHBDMA0_CTL0_L/H, and AHBDMA0_CFG0_L/H for multiblock DMA transfers using block chaining.

> **Note:** For rows 6 through 10 of *Table 21-2*, the LLI.AHBDMA0_CTL0_L/H, LLI.AHBDMA0_LLP0, LLI.AHBDMA0_SAR0, and LLI.AHBDMA0_DAR0 register locations of the LLI are always affected at the start of every block transfer. The LLI.AHBDMA0_LLP0 and LLI.AHBDMA0_CTL0_L/H locations are always used to reprogram the AHBDMA0_LLP0 and AHBDMA0_CTL0_L/H registers.

All register names in the following table are preceded by AHBDMA0_. So the complete register name for programming purposes is AHBDMA0_*regname* (e.g., AHBDMA0_CTL0_L/H).

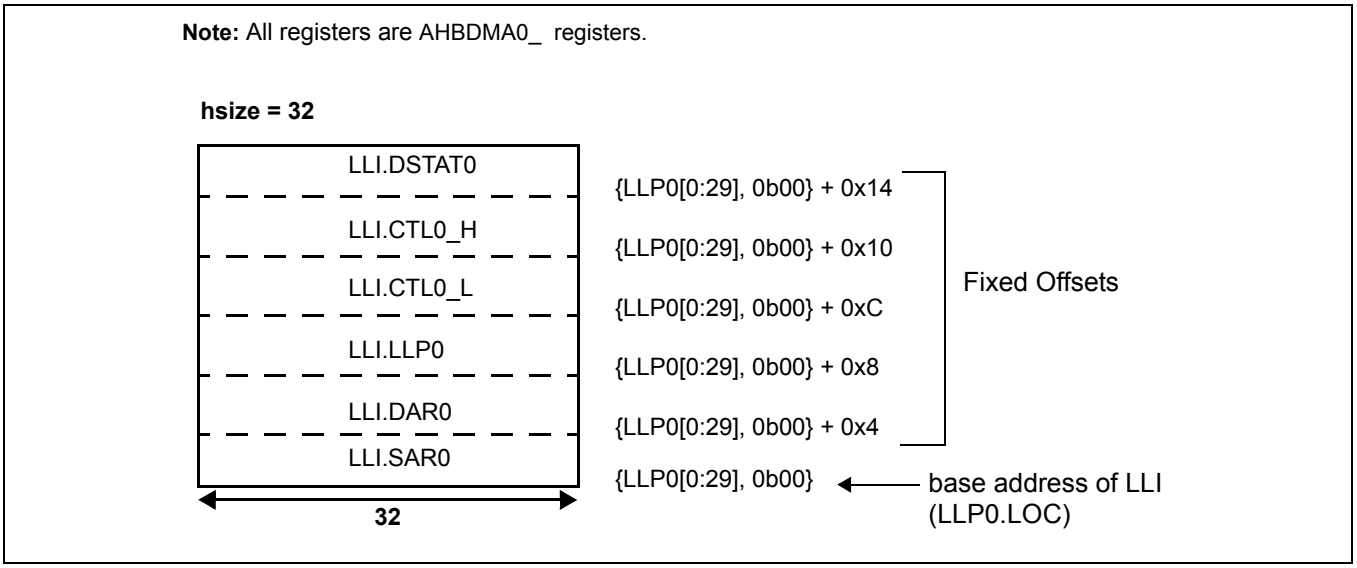
Table 21-2. Programming of Transfer Types and Channel Register (AHBDMA0_)Update Method

Transfer Type	LLP. LOC = 0	LLP_ SRC_EN (CTL0_L)	Reload _SRC (CFG0_L)	LLP_ DST_EN (CTL0_L)	Reload _DST (CFG0_L)	CTL0_L/H, LLP0 Update Method	SAR0 Update Method	DAR0 Update Method
1. Single-block or last transfer of multiblock.	Yes	0	0	0	0	None, user reprograms	None (single)	None (single)

Table 21-2. Programming of Transfer Types and Channel Register (AHBDMA0_)Update Method

2. Auto- reload multiblock transfer with contiguous SAR	Yes	0	0	0	1	CTL0_L/H, LLP0 are reloaded from initial values.	Contiguous	Auto Reload
3. Auto- reload multiblock transfer with contiguous DAR.	Yes	0	1	0	0	CTL0_L/H, LLP0 are reloaded from initial values	Auto Reload	Contiguous
4. Auto-reload multiblock transfer	Yes	0	1	0	1	CTL0_L/H, LLP0 are reloaded from initial values	Auto Reload	Auto Reload
5. Single-block or last transfer of multiblock.	No	0	0	0	0	None, user reprograms	None (single)	None (single)
6. Linked list multiblock transfer with contiguous SAR	No	0	0	1	0	CTL0_L/H, LLP0 loaded from next Linked List item.	Contiguous	Linked List
7. Linked list multiblock transfer with auto-reload SAR	No	0	1	1	0	CTL0_L/H, LLP0 loaded from next Linked List item.	Auto Reload	Linked List
8. Linked list multiblock transfer with contiguous DAR	No	1	0	0	0	CTL0_L/H, LLP0 loaded from next Linked List item.	Linked List	Contiguous
9. Linked list multiblock transfer with auto-reload DAR	No	1	0	0	1	CTL0_L/H, LLP0 loaded from next Linked List item.	Linked List	Auto Reload
10. Linked list multiblock transfer	No	1	0	1	0	CTL0_L/H, LLP0 loaded from next Linked List item.	Linked List	Linked List

Figure 21-43. Mapping of Block Descriptor (LLI) in Memory to Channel Registers



Note: Throughout this book, there are descriptions about fetching the LLI.AHBDMA0_CTL0_L/H register from the location pointed to by the AHBDMA0_LLP0 register. This exact location is the LLI base address (stored in AHBDMA0_LLP0 register) plus the fixed offset. For example, in the above figure, the location of the LLI.AHBDMA0_CTL0_L/H register is LLP0.AHBDMA0_LOC + 0xC.

21.13.3.2 Auto-Reloading of Channel Registers

During auto-reloading, the channel registers are reloaded with their initial values at the completion of each block and the new values used for the new block. Depending on the row number in *Table 21-2*, some or all of the AHBDMA0_SAR0, AHBDMA0_DAR0, and AHBDMA0_CTL0_L/H channel registers are reloaded from their initial value at the start of a block transfer.

21.13.3.3 Contiguous Address Between Blocks

In this case, the address between successive blocks is selected as a continuation from the end of the previous block.

Enabling the source or destination address to be contiguous between blocks is a function of the AHBDMA0_CTL0_L[LLP_SRC_EN], AHBDMA0_CFG0_L[RELOAD_SRC], AHBDMA0_CTL0_L[LLP_DST_EN], and AHBDMA0_CTL0_L[RELOAD_DST] fields. See *Table 21-2*.

You cannot select both AHBDMA0_SAR0 and AHBDMA0_DAR0 updates to be contiguous. If you want this functionality, you should increase the size of the Block Transfer (AHBDMA0_CTL0_L[BLOCK_TS]), or if this is at the maximum value, use Row 10 of *Table 21-2* and set up the LLI.AHBDMA0_SAR0 address of the block descriptor to be equal to the end AHBDMA0_SAR0 address of the previous block. Similarly, set up the LLI.AHBDMA0_DAR0 address of the block descriptor to be equal to the end AHBDMA0_DAR0 address of the previous block.

21.13.3.4 Suspension of Transfers Between Blocks

At the end of every block transfer, an end-of-block interrupt is asserted if:

1. Interrupts are enabled, AHBDMA0_CTL0_L[INT_EN] = 1, and
2. The channel block interrupt is unmasked, MaskBlock[31] = 1.

Note: The block-complete interrupt is generated at the completion of the block transfer to the destination.

For rows 6, 8, and 10 of *Table 21-2*, the DMA transfer does not stall between block transfers. For example, at the end-of-block N, the SATA DMA automatically proceeds to block N + 1.

For rows 2, 3, 4, 7, and 9 of *Table 21-2* (AHBDMA0_SAR0 and/or AHBDMA0_DAR0 auto-reloaded between block transfers), the DMA transfer automatically stalls after the end-of-block interrupt is asserted, if the end-of-block interrupt is enabled and unmasked.

The SATA DMA does not proceed to the next block transfer until a write to the ClearBlock[31] block interrupt clear register, done by software to clear the channel block-complete interrupt, is detected by hardware.

For rows 2, 3, 4, 7, and 9 of *Table 21-2* (AHBDMA0_SAR0 and/or AHBDMA0_DAR0 auto-reloaded between block transfers), the DMA transfer does not stall if either:
- Interrupts are disabled, AHBDMA0_CTL0_L[INT_EN] = 0, or
- The channel block interrupt is masked, MaskBlock[31] = 0.

Channel suspension between blocks is used to ensure that the end-of-block ISR (interrupt service routine) of the next-to-last block is serviced before the start of the final block commences. This ensures that the ISR has cleared the AHBDMA0_CFG0_L[RELOAD_SRC] and/or AHBDMA0_CFG0_L[RELOAD_DST] bits before completion of the final block. The reload bits AHBDMA0_CFG0_L[RELOAD_SRC] and/or AHBDMA0_CFG0_L[RELOAD_DST] should be cleared in the end-of-block ISR for the next-to-last block transfer.

21.13.3.5 Ending Multiblock Transfers

All multiblock transfers must end as shown in either Row 1 or Row 5 of *Table 21-2*. At the end of every block transfer, the SATA DMA samples the row number, and if the SATA DMA is in the Row 1 or Row 5 state, then the previous block transferred was the last block and the DMA transfer is terminated.

For rows 2, 3, and 4 of *Table 21-2*, (LLP0 = 0 and AHBDMA0_CFG0_L[RELOAD_SRC] and/or AHBDMA0_CFG0_L[RELOAD_DST] is set), multiblock DMA transfers continue until both AHBDMA0_CFG0_L[RELOAD_SRC] and AHBDMA0_CFG0_L[RELOAD_DST] are cleared by software. They should be programmed to 0 in the end-of-block interrupt service routine that services the next-to-last block transfer; this puts the SATA DMA into the Row 1 state.

For rows 6, 8, and 10 of *Table 21-2* (both AHBDMA0_CFG0_L[RELOAD_SRC] and AHBDMA0_CFG0_L[RELOAD_DST] cleared), the user must set up the last block descriptor in memory so that both LLI.AHBDMA0_CTL0_L.LLP_SRC_EN and LLI.AHBDMA0_CTL0_L.LLP_DST_EN are 0. If the LLI.AHBDMA0_LLP0 register of the last block descriptor in memory is non-zero, then the DMA transfer is terminated in Row 5. If the LLI.AHBDMA0_LLP0 register of the last block descriptor in memory is 0, then the DMA transfer is terminated in Row 1.

For Rows 7 and 9, the end-of-block interrupt service routine that services the next-to-last block transfer should clear the AHBDMA0_CFG0_L[RELOAD_SRC] and AHBDMA0_CFG0_L[RELOAD_DST] reload bits. The last block descriptor in memory should be set up so that both LLI.AHBDMA0_CTL0_L[LLP_SRC_EN] and LLI.AHBDMA0_CTL0_L[LLP_DST_EN] are 0. If the LLI.AHBDMA0_LLP0 register of the last block descriptor in memory is non-zero, then the DMA transfer is terminated in Row 5. If the LLI.AHBDMA0_LLP0 register of the last block descriptor in memory is 0, then the DMA transfer is terminated in Row 1.

Note: The only allowed transitions between the rows of *Table 21-2* are from any row into Row 1 or Row 5. As already stated, a transition into Row 1 or Row 5 is used to terminate the DMA transfer; all other transitions between rows are not allowed. Software must ensure that illegal transitions between rows do not occur between blocks of a multiblock transfer. For example, if block N is in Row 10, then the only allowed rows for block N +1 are Rows 10, 5, or 1.

21.13.4 Programming a Channel

Five registers – AHBDMA0_LLP0, AHBDMA0_CTL0L/H, and AHBDMA0_CFG0L/H – need to be programmed to determine whether single- or multiblock transfers occur, and which type of multiblock transfer is used. The different transfer types are shown in *Table 21-2*.

The "Update Method" columns indicate where the values of AHBDMA0_SAR0, AHBDMA0_DAR0, AHBDMA0_CTL0_L/H, and AHBDMA0_LLP0 are obtained for the next block transfer when multiblock SATA DMA transfers are enabled.

Note: In *Table 21-2*, all other combinations of AHBDMA0_LLP0[LOC] = 0, AHBDMA0_CTL0_L[LLP_SRC_EN], AHBDMA0_CFG0_L[RELOAD_SRC], AHBDMA0_CTL0_L[LLP_DST_EN], and AHBDMA0_CFG0_L[RELOAD_DST] are illegal, and cause indeterminate or erroneous behavior.

21.13.5 Programming Examples

The following sections provide examples of programming different SATA DMA transfers.

21.13.5.1 Single-Block Transfer (Row 1)

Note: Unless otherwise indicated, all registers referred to are AHBDMA0_ registers.

1. Read the Channel Enable register to choose a free (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: ClearTfr, ClearBlock, ClearSrcTran, ClearDstTran, and ClearErr. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a. Write the starting source address in the SAR0 register.
   b. Write the starting destination address in the DAR0 register.
   c. Program CTL0_L/H and CFG0_L/H according to Row 1, as shown in *Table 21-2*. Program the LLP0 register with 0.
   d. Write the control information for the DMA transfer in the CTL0_L/H register. For example, in the register, you can program the following:
      (1) Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the CTL0_L/H register.
      (2) Set up the transfer characteristics, such as:
         - Transfer width for the source in the SRC_TR_WIDTH field.
         - Transfer width for the destination in the DST_TR_WIDTH field.
         - Source master layer in the SMS field where the source resides.
         - Destination master layer in the DMS field where the destination resides.
         - Incrementing/decrementing or fixed address for the source in the SINC field.

- Incrementing/decrementing or fixed address for the destination in the DINC field.

e. Write the channel configuration information into the CFG0_L/H register.

Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.

This step requires programming the HS_SEL_SRC/HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests. Writing a 1 activates the software handshaking interface to handle source and destination requests.

f. If gather is enabled and CTL0_L[SRC_GATHER_EN] is enabled), program the SGR0 register for channel 0.

g. If scatter is enabled and CTL0_L[DST_SCATTER_EN]), program the DSR0 register (refer to page 184) for channel 0.

4. After the SATA DMA-selected channel has been programmed, enable the channel by writing a 1 to the ChEnReg[CH_EN] bit. Ensure that bit 31 of the DmaCfgReg register is enabled.
5. Program the MRD/MWR fields of the SATA0_DBTSR register in the SATA Host interface with the same burst sizes programmed in SATA DMA.
6. Set the RXCHEN/TXCHEN bits of the SATA0_DMACR register in Host SATA depending on the transfer direction. When the data transfer is complete clear the RXCHEN/TXCHEN bits in the SATA0_DMACR register.
7. Once the transfer completes, hardware sets the interrupts and disables the channel. At this time, you can respond to either the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RawTfr[31]) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, the software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, ClearTfr[31], before the channel is enabled.

Note: All of the remaining examples are multiblock transfers (MBTs)

21.13.5.2 MBT with Linked List for Source and Destination (Row 10)

Note: Unless otherwise indicated, all registers referred to are AHBDMA0_ registers.

1. Read the Channel Enable register to choose a free (disabled) channel.
2. Set up the chain of Linked List Items (otherwise known as block descriptors) in memory. Write the control information in the LLI.CTL0_L register location of the block descriptor for each LLI in memory 0. For example, in the register, you can program the following:
   a. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the CTL0_L register.
   b. Set up the transfer characteristics, such as:
      (1) Transfer width for the source in the SRC_TR_WIDTH field. This is typically set to 32 bits.
      (2) Transfer width for the destination in the DST_TR_WIDTH field. This is typically set to 32 bits.
      (3) Source master layer in the SMS field where the source resides.
      (4) Destination master layer in the DMS field where the destination resides.
      (5) Incrementing/decrementing or fixed address for the source in the SINC field. If the SATA Host interface is the source, set to fixed address or "No change."
      (6) Incrementing/decrementing or fixed address for the destination in the DINC field. If the SATA Host interface is the source, set to fixed address or "No change."
3. Write the channel configuration information into the CFG0_L/H register.

Designate the handshaking interface type (hardware or software) for the source and destination peripherals. This is not required for memory.

This step requires programming the HS_SEL_SRC/HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests. Use hardware handshaking between the SATA Host interface and the SATA DMA.

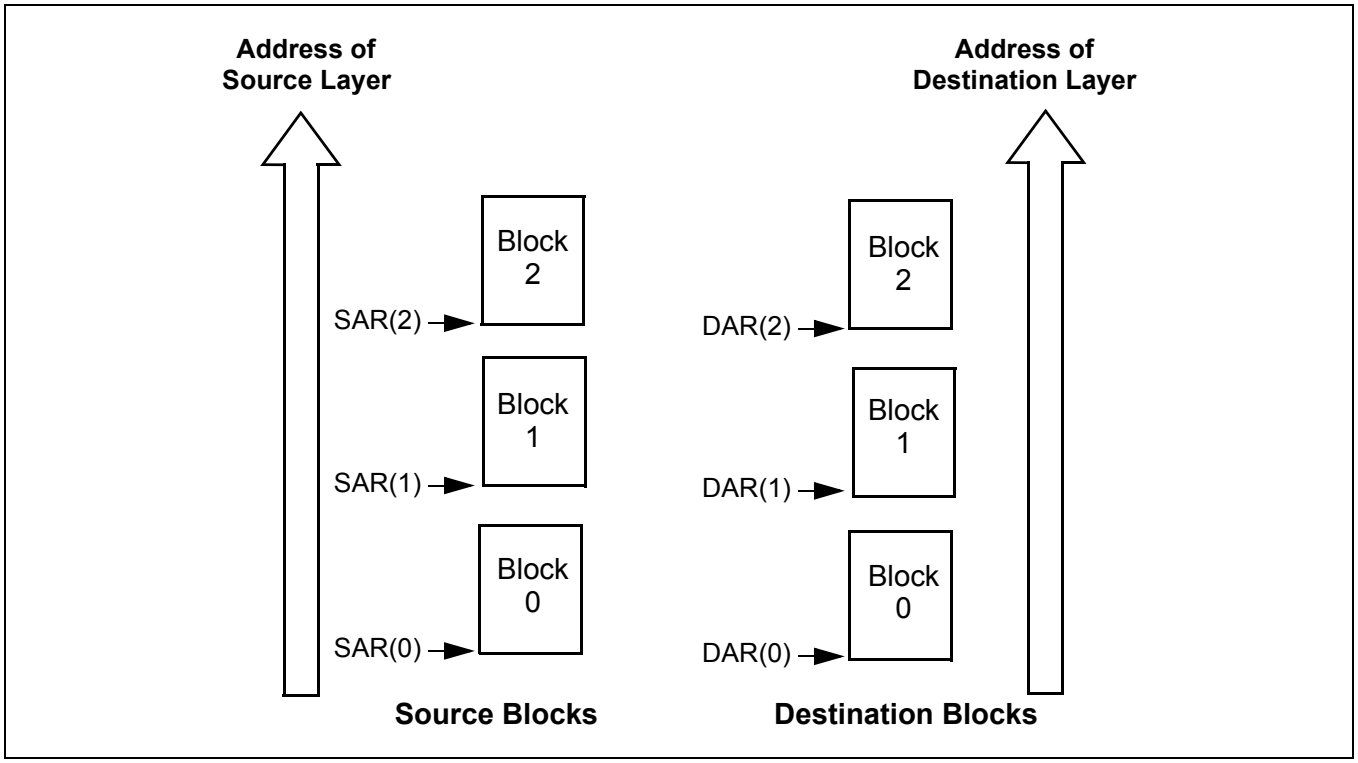
4. Make sure that the LLI.CTL0_L/H register locations of all LLI entries in memory (except the last) are set as shown in Row 10 of *Table 21-2*. The LLI.CTL0_L/H register of the last Linked List Item must be set as described in Row 1 or Row 5 of *Table 21-2*.
5. Make sure that the LLI.LLP0 register locations of all LLI entries in memory (except the last) are non-zero and point to the base address of the next Linked List Item.
6. Make sure that the LLI.SAR0/LLI.DAR0 register locations of all LLI entries in memory point to the start source/destination block address preceding that LLI fetch.
7. If gather is enabled (CTL0_L[SRC_GATHER_EN] is enabled), program the SGR0 register.
8. If scatter is enabled (CTL0_L[DST_SCATTER_EN] is enabled) program the DSR0 register.
9. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: ClearTfr, ClearBlock, ClearSrcTran, ClearDstTran, and ClearErr. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
10. Program the CTL0_L/H and CFG0_L/H registers according to Row 10, as shown in *Table 21-2*.
11. Program the LLP0 register with LLP(0), the pointer to the first linked list item.
12. Finally, enable the channel by writing a 1 to the ChEnReg[CH_EN] bit; the transfer is performed.
13. The SATA DMA fetches the first LLI from the location pointed to by LLP0(0).

    Note: The LLI.SAR0, LLI.DAR0, LLI.LLP0, and LLI.CTL0_L/H registers are fetched. The SATA DMA automatically reprograms the SAR0, DAR0, LLP0, and CTL0_L/H channel registers from the LLP0(0).
14. Source and destination request single and burst DMA transactions to transfer the block of data (assuming non-memory peripheral). The SATA DMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
15. The CTL0_H register is written out to the same location on the same layer (LLP0.LMS) where it was originally fetched; that is, the location of the CTL0_H register of the linked list item fetched prior to the start of the block transfer. The CTL0_H register is written out because the CTL0_H[BLOCK_TS] and CTL0_H[DONE] fields have been updated by the SATA DMA hardware. Additionally, the CTL0_H[DONE] bit is asserted to indicate block completion. Therefore, software can poll the LLI.CTL0_H[DONE] bit of the CTL0_H register in the LLI to know when a block transfer has completed.

    Note: Do not poll the CTL0_H[DONE] bit in the SATA DMA memory map. Instead, poll the LLI.CTL0_H[DONE] bit in the LLI for that block. If the polled LLI.CTL0_H[DONE] bit is asserted, then this block transfer has completed. This LLI.CTL0_H[DONE] bit was cleared at the start of the transfer.
16. The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
17. The SATA DMA does not wait for the block interrupt to be cleared, but continues fetching the next LLI from the memory location pointed to by the current LLP0 register and automatically reprograms the SAR0, DAR0, LLP0, and CTL0_L/H channel registers. The DMA transfer continues until the SATA DMA determines that the CTL0_L/H and LLP0 registers at the end of a block transfer match the ones described in Row 1 or Row 5 of *Table 21-2* (as discussed earlier). The SATA DMA then knows that the previously transferred block was the last block in the DMA transfer.

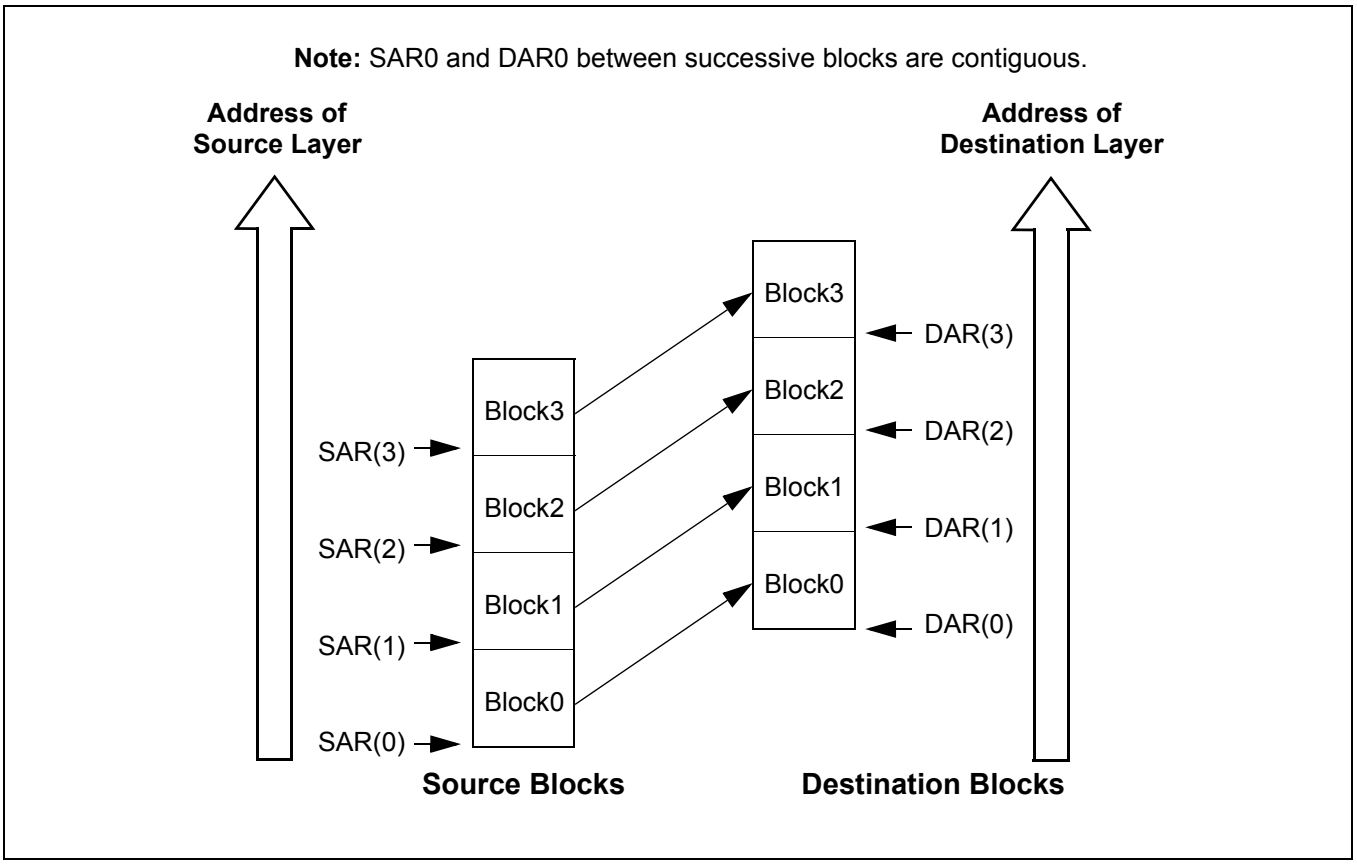
The DMA transfer might look like that shown in *Figure 21-44*.

Figure 21-44. MBT with Linked Address for Source and Destination



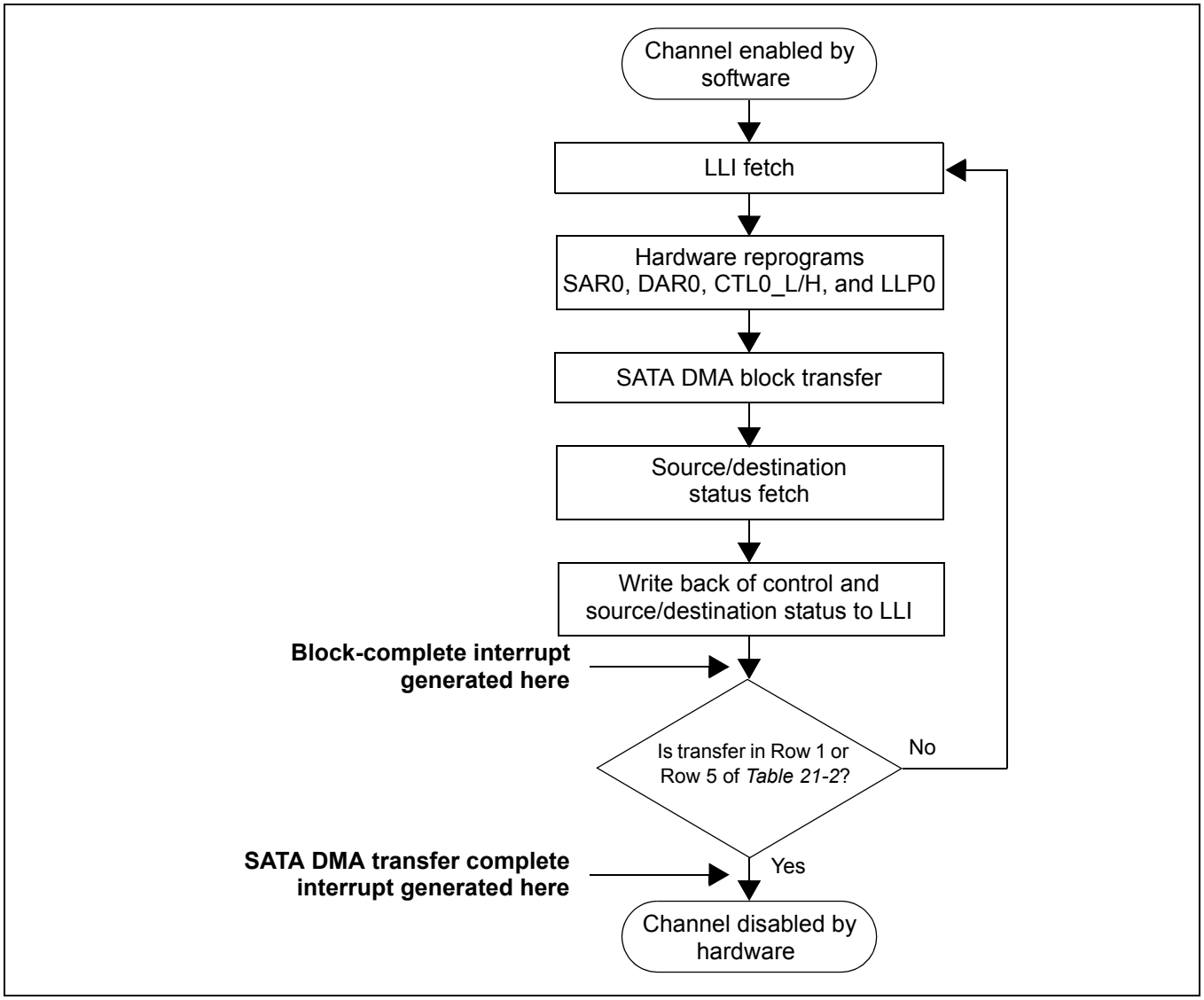
If the user needs to execute a DMA transfer where the source and destination address are contiguous, but where the amount of data to be transferred is greater than the maximum block size CTL0_H[BLOCK_TS], it can be achieved using the type of multiblock transfer shown in *Figure 21-45*.

Figure 21-45. MBT with Linked Address for Source and Destination



The DMA transfer flow is shown in the following figure.

Figure 21-46. Transfer Flow for Source and Destination Linked List Address



21.13.5.3 MBT with Source and Destination Address Auto-Reloaded (Row 4)

Note: Unless otherwise indicated, all registers referred to are AHBDMA0_ registers.

1. Read the Channel Enable register to choose an available (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: ClearTfr, ClearBlock, ClearSrcTran, ClearDstTran, and ClearErr. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a. Write the starting source address in the SAR0 register.
   b. Write the starting destination address in the DAR0 register.

c. Program CTL0_L/H and CFG0_L/H according to Row 4 as shown in *Table 21-2*. Program the LLP0 register with 0.

d. Write the control information for the DMA transfer in the CTL0_L/H register. For example, in the register, you can program the following:

   (1) Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the CTL0_L register.

   (2) Set up the transfer characteristics, such as:

   - Transfer width for the source in the SRC_TR_WIDTH field. Typically, this is 32 bits.
   - Transfer width for the destination in the DST_TR_WIDTH field. Typically, this is 32 bits.
   - Source master layer in the SMS field where the source resides.
   - Destination master layer in the DMS field where the destination resides.
   - Incrementing/decrementing or fixed address for the source in the SINC field. If the SATA Host interface is the source, set to fixed address or "No change."
   - Incrementing/decrementing or fixed address for the destination in the DINC field. If the SATA Host interface is the source, set to fixed address or "No change."

e. If gather is enabled (CTL0_L[SRC_GATHER_EN] is enabled), program the SGR0 register.

f. If scatter is enabled (CTL0_L[DST_SCATTER_EN]), program the DSR0 register.

g. Write the channel configuration information into the CFG0_L/H register. Ensure that the reload bits, CFG0_L[RELOAD_SRC] and CFG0_L[RELOAD_DST], are enabled.

   Designate the handshaking interface type (hardware or software) for the source and destination peripherals. This is not required for memory.

   This step requires programming the HS_SEL_SRC/HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests. Use hardware handshaking between the SATA Host interface and the SATA DMA.

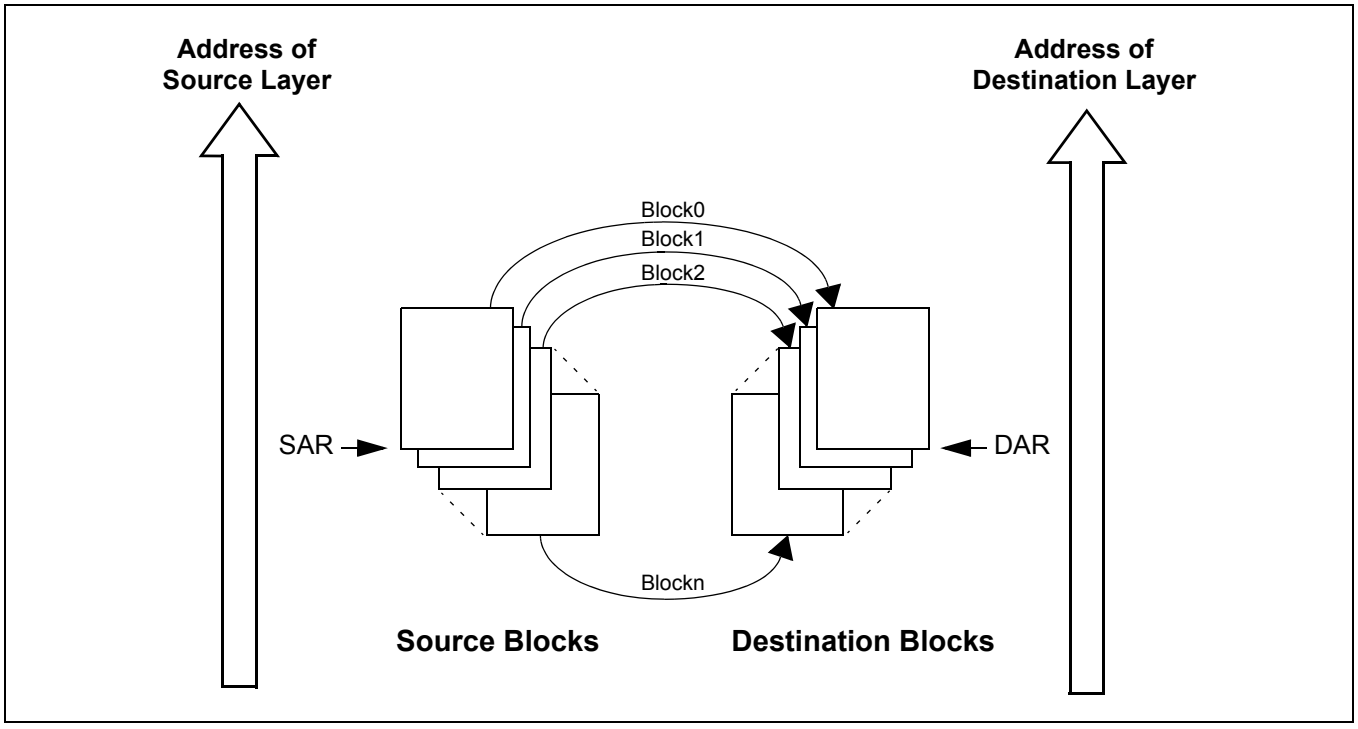
4. After the SATA DMA selected channel has been programmed, enable the channel by writing a 1 to the ChEnReg[CH_EN] bit. Ensure that bit 31 of the DmaCfgReg register is enabled.
5. Source and destination request single and burst SATA DMA transactions to transfer the block of data (assuming non-memory peripherals). The SATA DMA acknowledges on completion of each burst/single transaction and carries out the block transfer.
6. When the block transfer has completed, the SATA DMA reloads the SAR0, DAR0, and CTL0_L/H registers. Hardware sets the block-complete interrupt. The SATA DMA then samples the row number, as shown in *Table 21-2*. If the SATA DMA is in Row 1, then the DMA transfer has completed. Hardware sets the transfer complete interrupt and disables the channel. You can either respond to the Block Complete or Transfer Complete interrupts or poll for the transfer complete raw interrupt status register (RawTfr) until it is set by hardware to detect when the transfer is complete. Note that if polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, ClearTfr, before the channel is enabled. If the SATA DMA is not in Row 1, the next step is performed.
7. The DMA transfer proceeds as follows:

a. If interrupts are enabled (CTL0_L[INT_EN] = 1) and the block-complete interrupt is unmasked (MaskBlock = 1), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the reload bits in the CFG0_L.RELOAD_SRC and CFG0_L.RELOAD_DST registers. This puts the SATA DMA into Row 1, as shown in *Table 21-2*. If the next block is not the last block in the DMA transfer, then the reload bits should remain enabled to keep the SATA DMA in Row 4.

b. If interrupts are disabled (CTL0_L[INT_EN] = 0) or the block-complete interrupt is masked (MaskBlock = 0b0), then hardware does not stall until it detects a write to the block-complete interrupt clear register. Instead, it immediately starts the next block transfer. In this case, software must clear the reload bits in the CFG0_L[RELOAD_SRC] and CFG0_L[RELOAD_DST] registers to put the SATA DMA into Row 1 of *Table 21-2* before the last block of the DMA transfer has completed.

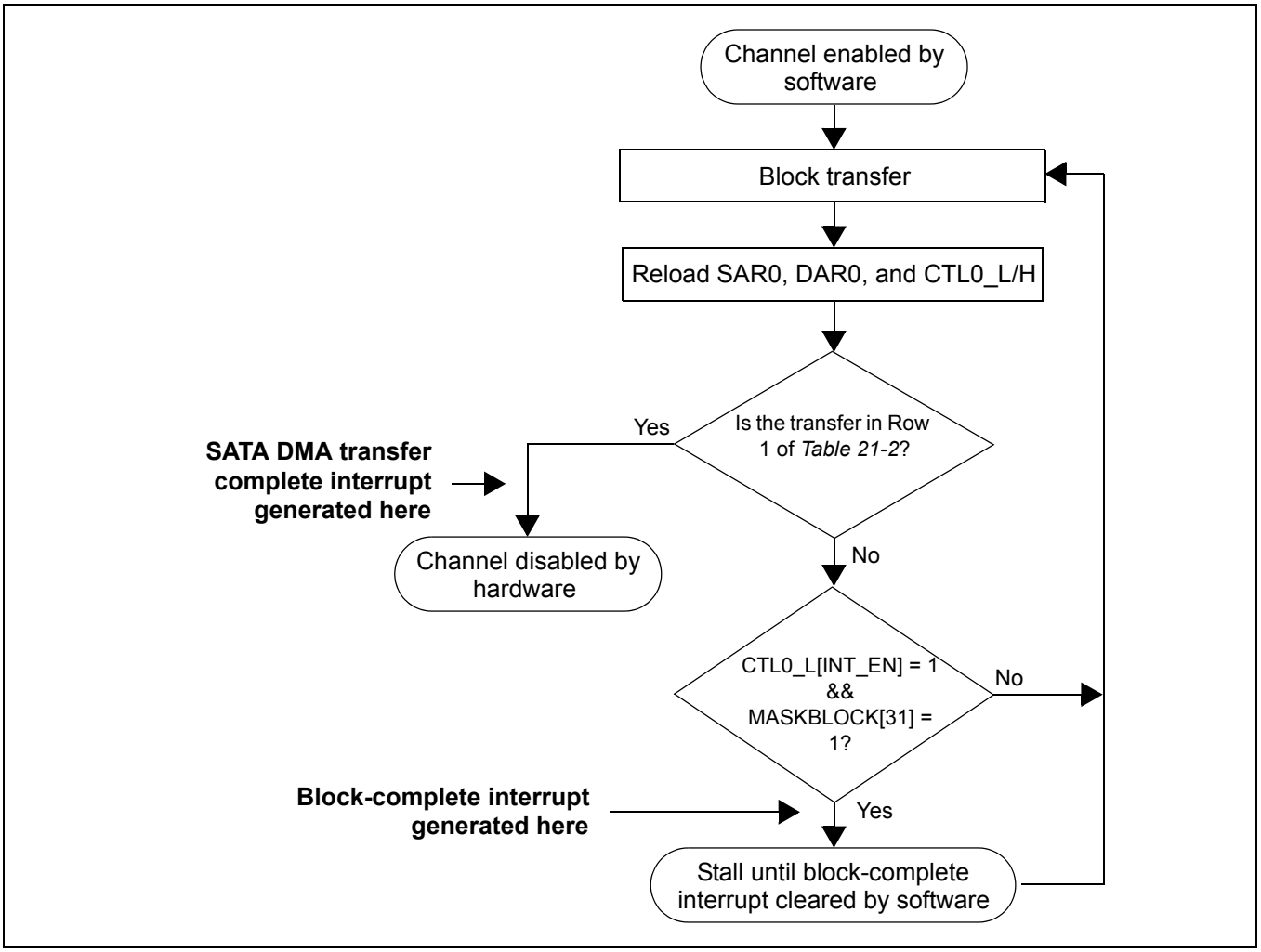
The transfer is similar to that shown in the following figure.

Figure 21-47. MBT with Source and Destination Address Auto-Reloaded



The DMA transfer flow is shown in the following figure.

Figure 21-48. MBT Flow for Source and Destination Address Auto-Reloaded



21.13.5.4 MBT with Source Address Auto-Reloaded and Linked List Destination Address (Row 7)

Note: Unless otherwise indicated, all registers referred to are AHBDMA0_ registers.

1. Read the Channel Enable register in order to choose a free (disabled) channel.
2. Set up the chain of linked list items (otherwise known as block descriptors) in memory. Write the control information in the LLI.CTL0_L/H register location of the block descriptor for each LLI in memory. For example, in the register you can program the following:
   a. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control peripheral by programming the TT_FC of the CTL0_L register.
   b. Set up the transfer characteristics, such as:
      (1) Transfer width for the source in the SRC_TR_WIDTH field. Typically, this is 32 bits.
      (2) Transfer width for the destination in the DST_TR_WIDTH field. Typically, this is 32 bits.
      (3) Source master layer in the SMS field where the source resides.

(4) Destination master layer in the DMS field where the destination resides.

(5) Incrementing/decrementing or fixed address for the source in the SINC field. If the SATA Host interface is the source, set to fixed address or "No change."

(6) Incrementing/decrementing or fixed address for the destination in the DINC field. If the SATA Host interface is the source, set to fixed address or "No change."

3. Write the starting source address in the SAR0 register.

   Note: The values in the LLI.SAR0 register locations of each of the Linked List Items (LLIs) set up in memory, although fetched during an LLI fetch, are not used.

4. Write the channel configuration information into the CFG0_L/H register.

   Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.

   This step requires programming the HS_SEL_SRC/HS_SEL_DST bits. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface source/destination requests. Use hardware handshaking between the SATA Host interface and the SATA DMA. Use hardware handshaking between the SATA Host interface and the SATA DMA.

5. Make sure that the LLI.CTL0_L/H register locations of all LLIs in memory (except the last) are set as shown in Row 7 of *Table 21-2*, while the LLI.CTL0_L/H register of the last Linked List item must be set as described in Row 1 or Row 5 of *Table 21-2*.

6. Ensure that the LLI.LLP0 register locations of all LLIs in memory (except the last) are non-zero and point to the next Linked List Item.

7. Ensure that the LLI.DAR0 register location of all LLIs in memory point to the start destination block address preceding that LLI fetch.

8. If gather is enabled (CTL0_L.SRC_GATHER_EN is enabled), program the SGR0 register.

9. If scatter is enabled (CTL0_L.DST_SCATTER_EN0) program the DSR0 register.

10. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: ClearTfr, ClearBlock, ClearSrcTran, ClearDstTran, and ClearErr. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.

11. Program the CTL0_L/H and CFG0_L/H registers according to Row 7, as shown in *Table 21-2*

12. Program the LLP0 register with LLP0(0), the pointer to the first Linked List item.

13. Finally, enable the channel by writing a 1 to the ChEnReg.CH_EN bit; the transfer is performed. Ensure that bit 31 of the DmaCfgReg register is enabled.

14. The SATA DMA fetches the first LLI from the location pointed to by LLP0(0).

    Note: The LLI.SAR0, LLI.DAR0, LLI.LLP0, and LLI.CTL0_L/H registers are fetched. The LLI.SAR0 register, although fetched, is not used.

15. Source and destination request single and burst SATA DMA transactions in order to transfer the block of data (assuming non-memory peripherals). The SATA DMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.

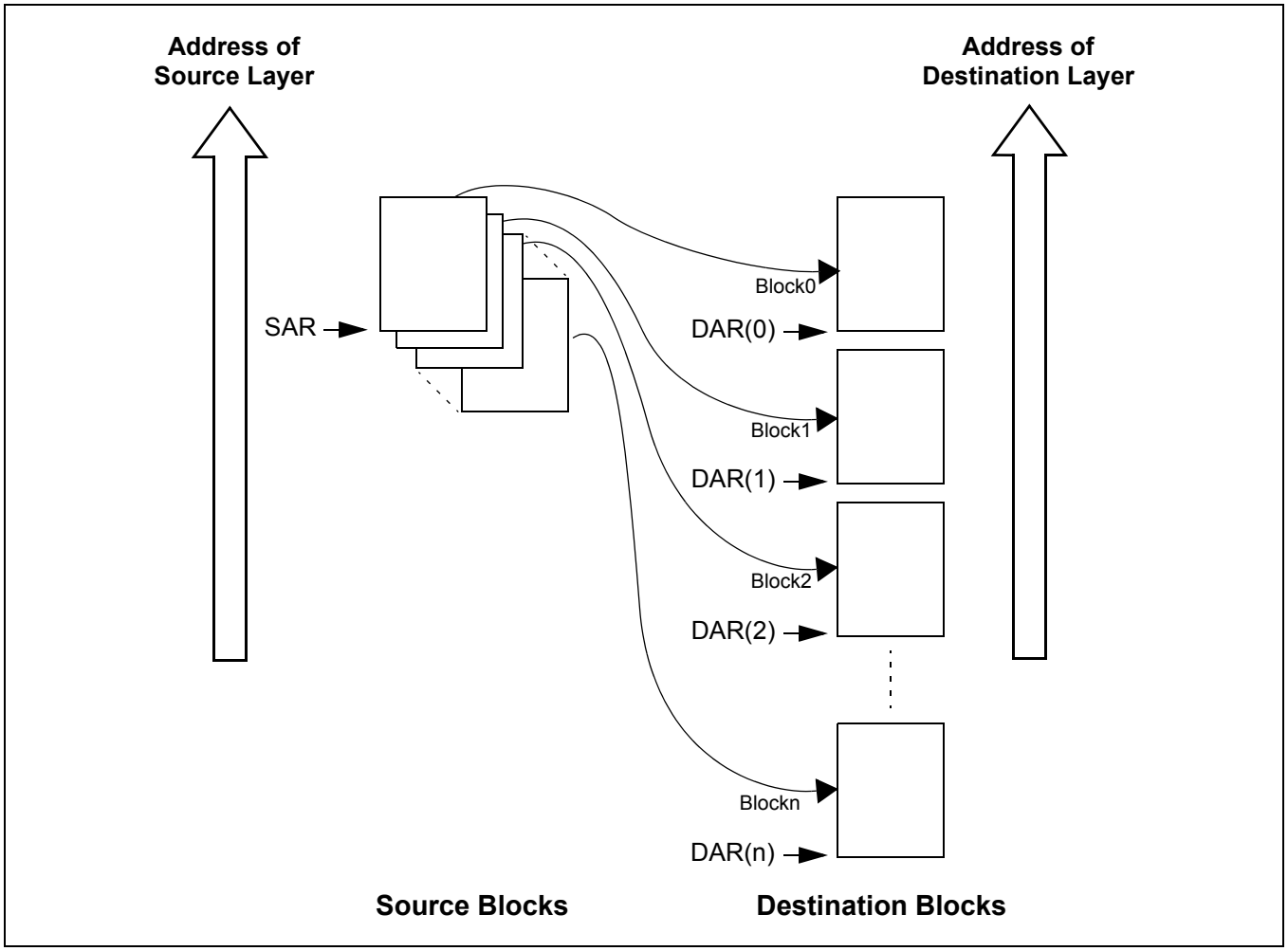
16. The CTL0_H register is written out to the same location on the same layer (LLP0[LMS]) where it was originally fetched; that is, the location of the CTL0_H register of the linked list item fetched prior to the start of the block transfer. CTL0_H is written out because the CTL0_H[BLOCK_TS] and CTL0_H[DONE] fields have been updated by hardware within the SATA DMA. The LLI.CTL0_H[DONE] bit is asserted to indicate block completion. Therefore, software can poll the LLI.CTL0_H[DONE] bit field of the CTL0_H register in the LLI to know when a block transfer has completed.

Note: Do not poll the CTL0_H[DONE] bit in the SATA DMA memory map. Instead, poll the LLI.CTL0_H[DONE] bit in the LLI for that block. If the polled LLI.CTL0_H[DONE] bit is asserted, then this block transfer has completed. This LLI.CTL0_H[DONE] bit was cleared at the start of the transfer.

17. The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
18. The SATA DMA reloads the SAR0 register from the initial value. Hardware sets the block-complete interrupt. The SATA DMA samples the row number, as shown in *Table 21-2*. If the SATA DMA is in Row 1 or Row 5, then the DMA transfer has completed. Hardware sets the transfer complete interrupt and disables the channel. You can either respond to the Block Complete or Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RawTfr) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, ClearTfr, before the channel is enabled. If the SATA DMA is not in Row 1 or Row 5 as shown in *Table 21-2*, the following steps are performed.
19. The DMA transfer proceeds as follows:
    a. If interrupts are enabled (CTL0_L[INT_EN] = 1) and the block-complete interrupt is unmasked (MaskBlock = 1), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the CFG0_L[RELOAD_SRC] source reload bit. This puts the SATA DMA into Row 1, as shown in *Table 21-2*. If the next block is not the last block in the DMA transfer, then the source reload bit should remain enabled to keep the SATA DMA in Row 7, as shown in *Table 21-2*.
    b. If interrupts are disabled (CTL0_L[INT_EN] = 0) or the block-complete interrupt is masked (MaskBlock = 0, then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it immediately starts the next block transfer. In this case, software must clear the source reload bit, CFG0_L[RELOAD_SRC], in order to put the device into Row 1 of *Table 21-2* before the last block of the DMA transfer has completed.
20. The SATA DMA fetches the next LLI from memory location pointed to by the current LLP0 register and automatically reprograms the DAR0, CTL0_L/H, and LLP0 channel registers. Note that the SAR0 is not reprogrammed, since the reloaded value is used for the next DMA block transfer. If the next block is the last block of the DMA transfer, then the CTL0_L/H and LLP0 registers just fetched from the LLI should match Row 1 or Row 5 of *Table 21-2*.

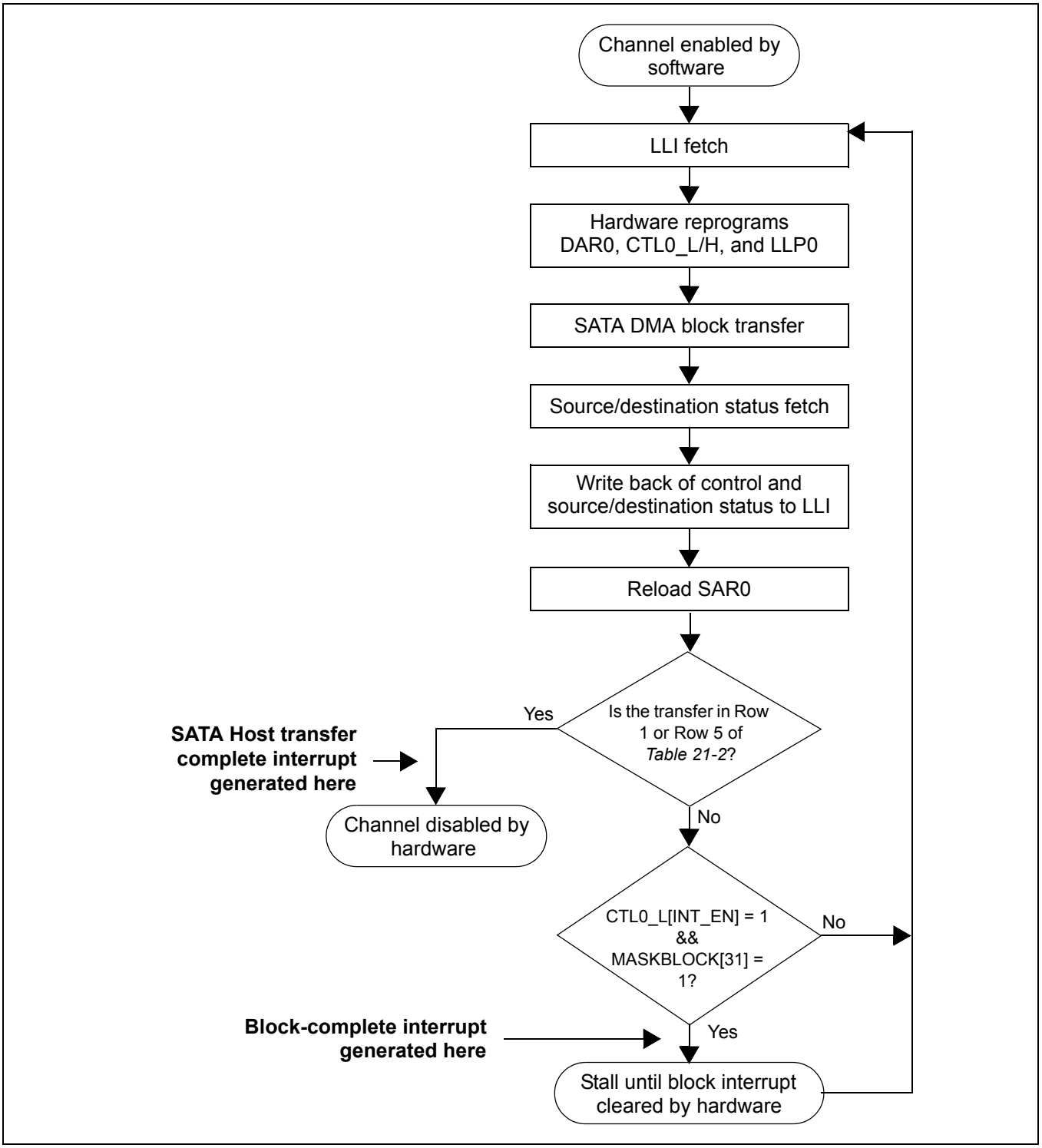
The DMA transfer might look like that shown in the following figure.

Figure 21-49. MBT with Source Address Auto-Reloaded and Linked List Destination Address



The DMA transfer flow is shown in the following figure.

Figure 21-50. MBT Flow for Source Address Auto-Reloaded and Linked List Destination Address



21.13.5.5 MBT with Source Address Auto-Reloaded and Contiguous Destination Address (Row 3)

Note: Unless otherwise indicated, all registers referred to are AHBDMA0_ registers.

1. Read the Channel Enable register to choose a free (disabled) channel.
2. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: ClearTfr, ClearBlock, ClearSrcTran, ClearDstTran, and ClearErr. Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
3. Program the following channel registers:
   a. Write the starting source address in the SAR0 register.
   b. Write the starting destination address in the DAR0 register.
   c. Program CTL0_L/H and CFG0_L/H according to Row 3, shown in *Table 21-2*. Program the LLP0 register with 0.
   d. Write the control information for the DMA transfer in the CTL0_L register. For example, in the register, you can program the following:
      (1) Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the CTL0_L register.
      (2) Set up the transfer characteristics, such as:
         - Transfer width for the source in the SRC_TR_WIDTH field. Typically, this is 32 bits.
         - Transfer width for the destination in the DST_TR_WIDTH field. Typically, this is 32 bits.
         - Source master layer in the SMS field where the source resides.
         - Destination master layer in the DMS field where the destination resides.
         - Incrementing/decrementing or fixed address for the source in the SINC field. If the SATA Host interface is the source, set to fixed address or "No change."
         - Incrementing/decrementing or fixed address for the destination in the DINC field. If the SATA Host interface is the source, set to fixed address or "No change."
   e. If gather is enabled (CTL0_L[SRC_GATHER_EN] is enabled), program the SGR0 register 0.
   f. If scatter is enabled (CTL0_L[DST_SCATTER_EN] is enabled), program the DSR0 register.
   g. Write the channel configuration information into the CFG0_L/H register.

      Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.

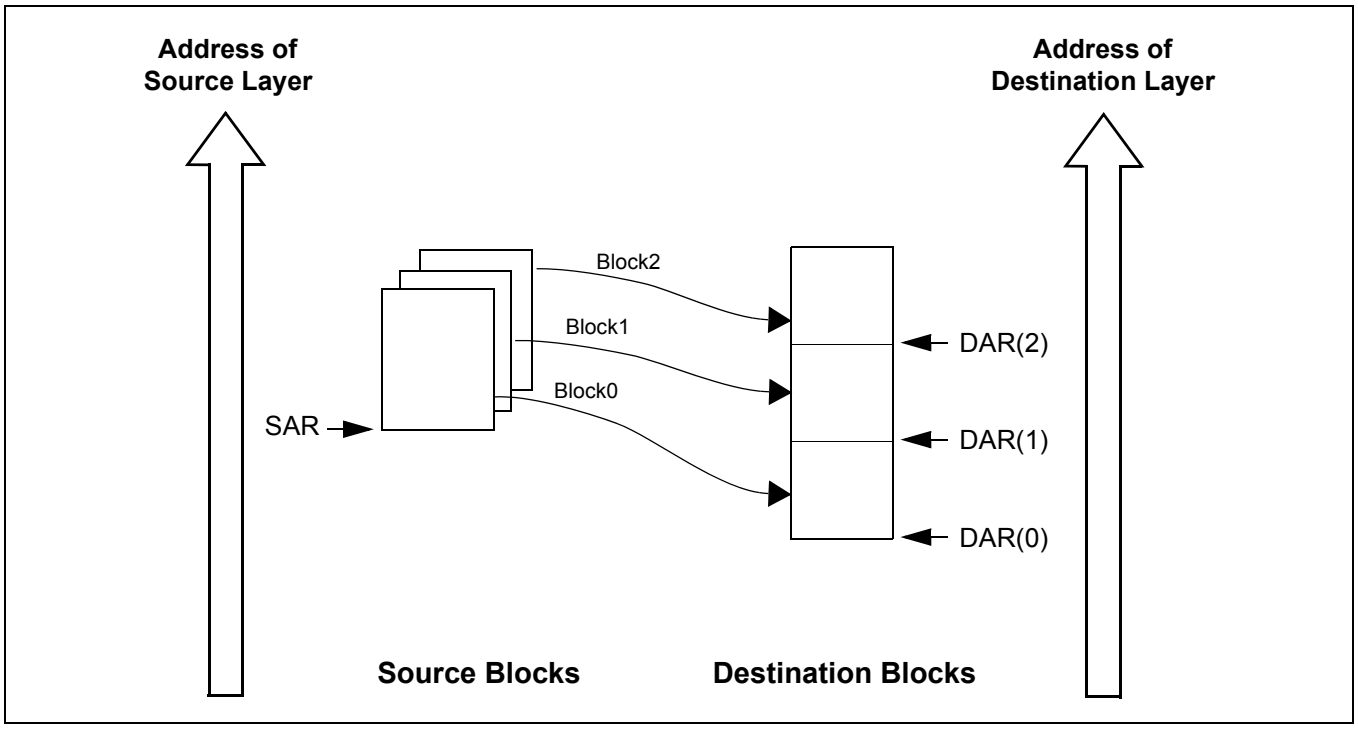
      This step requires programming the HS_SEL_SRC/HS_SEL_DST bits, respectively. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests. Use hardware handshaking between the SATA Host interface and the SATA DMA.
4. After the SATA DMA channel has been programmed, enable the channel by writing a 1 to the ChEnReg[CH_EN] bit. Ensure that bit 31 of the DmaCfgReg register is enabled.
5. Source and destination request single and burst SATA DMA transactions to transfer the block of data (assuming non-memory peripherals). The SATA DMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
6. When the block transfer has completed, the SATA DMA reloads the SAR0 register; the DAR0 register remains unchanged. Hardware sets the block-complete interrupt. The SATA DMA then samples the row number, as shown in *Table 21-2*. If the SATA DMA is in Row 1, then the DMA transfer has completed. Hardware sets the transfer-complete interrupt and disables the channel. You can either respond to the Block Complete or

Transfer Complete interrupts, or poll for the transfer complete raw interrupt status register (RawTfr[31]) until it is set by hardware, in order to detect when the transfer is complete. Note that if this polling is used, software must ensure that the transfer complete interrupt is cleared by writing to the Interrupt Clear register, ClearTfr[31], before the channel is enabled. If the SATA DMA is not in Row 1, the next step is performed.

7. The DMA transfer proceeds as follows:
   a. If interrupts are enabled (CTL0_L[INT_EN] = 1) and the block-complete interrupt is unmasked (MaskBlock[31] = 1), hardware sets the block-complete interrupt when the block transfer has completed. It then stalls until the block-complete interrupt is cleared by software. If the next block is to be the last block in the DMA transfer, then the block-complete ISR (interrupt service routine) should clear the source reload bit, CFGx.RELOAD_SRC. This puts the SATA DMA into Row 1, as shown in *Table 21-2*. If the next block is not the last block in the DMA transfer, then the source reload bit should remain enabled to keep the SATA DMA in Row 3, as shown in *Table 21-2*.
   b. If interrupts are disabled (CTL0_L[INT_EN] = 0) or the block-complete interrupt is masked (MaskBlock[31] = 0), then hardware does not stall until it detects a write to the block-complete interrupt clear register; instead, it starts the next block transfer immediately. In this case, software must clear the source reload bit, CFG0_L[RELOAD_SRC], to put the device into Row 1 of *Table 21-2* before the last block of the DMA transfer has completed.

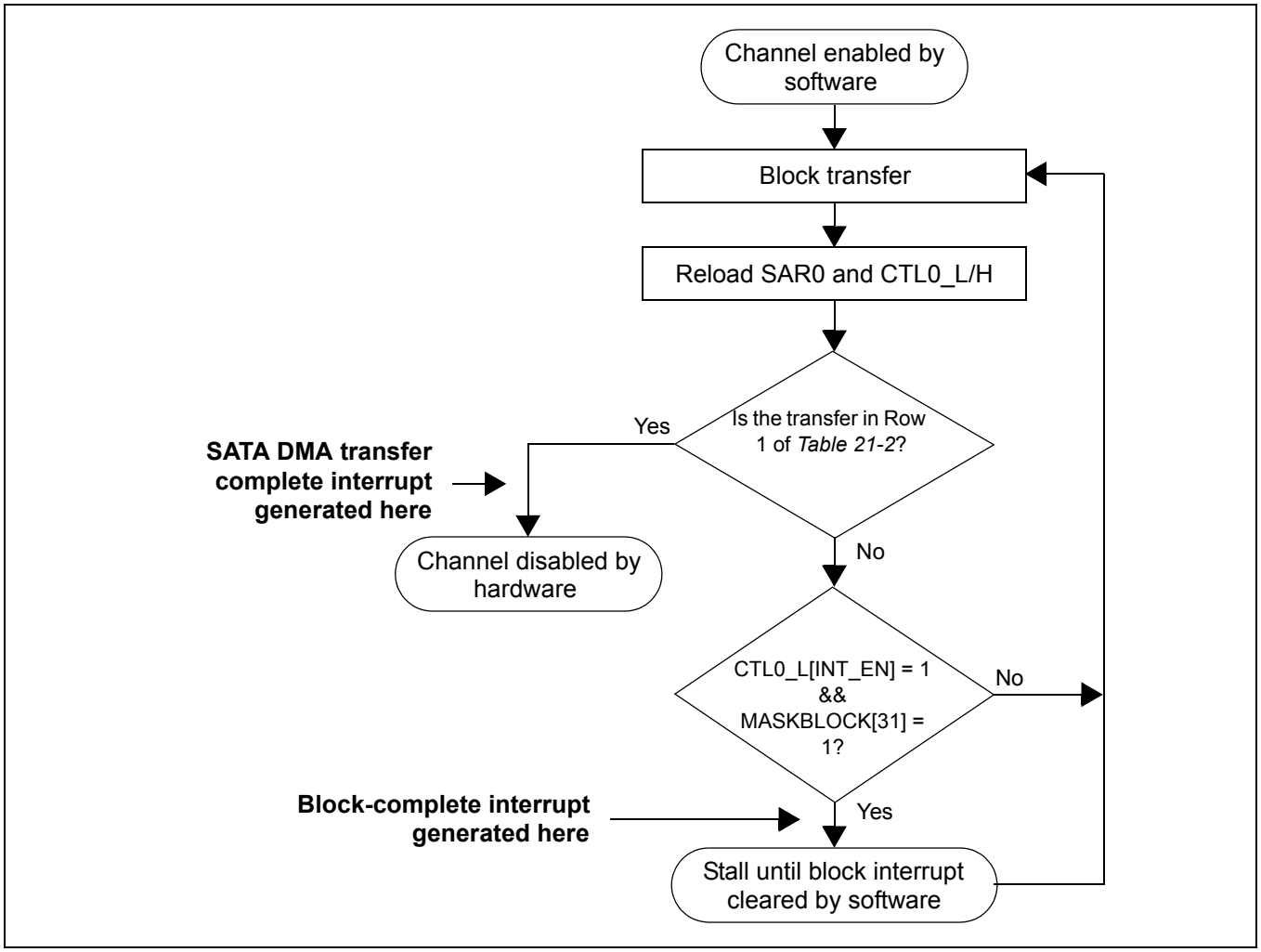
The DMA transfer flow is similar to that shown in the following figure.

Figure 21-51. MBT with Source Address Auto-Reloaded and Contiguous Destination Address



The DMA transfer flow is shown in the following figure.

Figure 21-52. MBT Flow for Source Address Auto-Reloaded and Contiguous Destination Address



21.13.5.6 MBT with Linked List for Source and Contiguous Destination Address (Row 8)

Note: Unless otherwise indicated, all registers referred to are AHBDMA0_ registers.

1. Read the Channel Enable register to choose a free (disabled) channel.
2. Set up the linked list in memory. Write the control information in the LLI.CTL0x register location of the block descriptor for each LLI in memory f. For example, in the register, you can program the following:
   a. Set up the transfer type (memory or non-memory peripheral for source and destination) and flow control device by programming the TT_FC of the CTL0 register.
   b. Set up the transfer characteristics, such as:
      (1) Transfer width for the source in the SRC_TR_WIDTH field. Typically, this is 32 bits.
      (2) Transfer width for the destination in the DST_TR_WIDTH field. Typically, this is 32 bits.
      (3) Source master layer in the SMS field where the source resides.
      (4) Destination master layer in the DMS field where the destination resides.

(5) Incrementing/decrementing or fixed address for the source in the SINC field. If the SATA Host interface is the source, set to fixed address or "No change."

(6) Incrementing/decrementing or fixed address for the destination in the DINC field. If the SATA Host interface is the source, set to fixed address or "No change."

3. Write the starting destination address in the DAR0 register.

   Note: The values in the LLI.DAR0 register location of each Linked List Item (LLI) in memory, although fetched during an LLI fetch, are not used.

4. Write the channel configuration information into the CFG0_L/H register.

   Designate the handshaking interface type (hardware or software) for the source and destination peripherals; this is not required for memory.

   This step requires programming the HS_SEL_SRC/HS_SEL_DST bits. Writing a 0 activates the hardware handshaking interface to handle source/destination requests for the specific channel. Writing a 1 activates the software handshaking interface to handle source/destination requests. Use hardware handshaking between the SATA Host interface and the SATA DMA.

5. Ensure that all LLI.CTL0_L/H register locations of the LLI (except the last) are set as shown in Row 8 of *Table 21-2*, while the LLI.CTL0_L/H register of the last Linked List item must be set as described in Row 1 or Row 5 of *Table 21-2*.
6. Ensure that the LLI.LLP0 register locations of all LLIs in memory (except the last) are non-zero and point to the next Linked List Item.
7. Ensure that the LLI.SAR0 register location of all LLIs in memory point to the start source block address preceding that LLI fetch.
8. If gather is enabled (CTL0_L[SRC_GATHER_EN] is enabled), program the SGR0 register.
9. If scatter is enabled (CTL0_L[DST_SCATTER_EN]) program the DSR0 register.
10. Clear any pending interrupts on the channel from the previous DMA transfer by writing to the Interrupt Clear registers: ClearTfr, ClearBlock, ClearSrcTran, ClearDstTran, and ClearErr . Reading the Interrupt Raw Status and Interrupt Status registers confirms that all interrupts have been cleared.
11. Program the CTL0_L/H and CFG0_L/H registers according to Row 8, as shown in *Table 21-2*.
12. Program the LLP0 register with LLP0(0), the pointer to the first Linked List item.
13. Finally, enable the channel by writing a 1 to the ChEnReg[CH_EN] bit; the transfer is performed. Ensure that bit 0 of the DmaCfgReg register is enabled.
14. The SATA DMA fetches the first LLI from the location pointed to by LLP0(0).

    Note: The LLI.SAR0, LLI.DAR0, LLI.LLP0, and LLI.CTL0_L/H registers are fetched. The LLI.DAR0 register location of the LLI – although fetched – is not used. The DAR0 register in the SATA DMA remains unchanged.

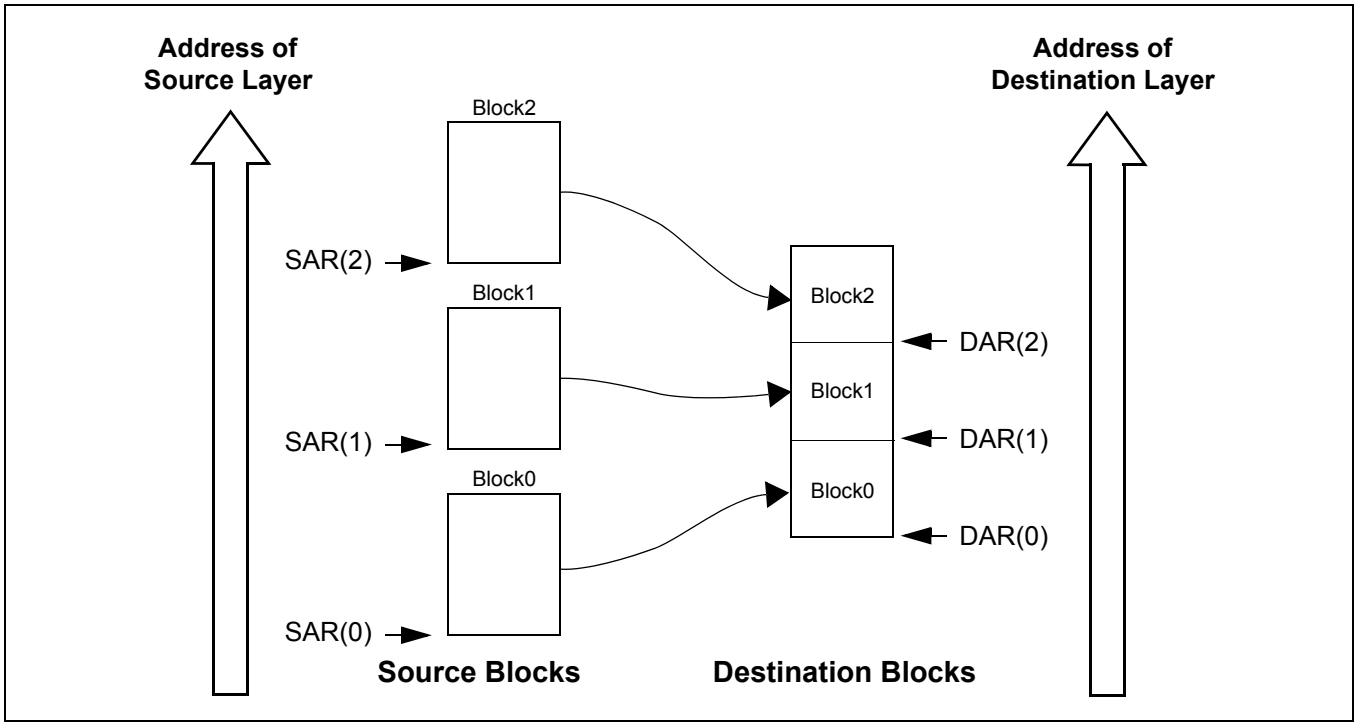
15. Source and destination request single and burst SATA DMA transactions to transfer the block of data (assuming non-memory peripherals). The SATA DMA acknowledges at the completion of every transaction (burst and single) in the block and carries out the block transfer.
16. The CTL0_H register is written out to the same location on the same layer (LLP0[LMS]) where it was originally fetched; that is, the location of the CTL0_H register of the linked list item fetched prior to the start of the block transfer. CTL0_H is written out because the CTL0_H[BLOCK_TS] and CTL0_H[DONE] fields have been updated by hardware within the SATA DMA. Additionally, the CTL0_H[DONE] bit is asserted to indicate block completion. Therefore, software can poll the LLI.CTL0_H[DONE] bit field of the CTL0_H register in the LLI to know when a block transfer has completed.

Note: Do not poll the CTL0_L/H[DONE] bit in the SATA DMA memory map. Instead, poll the LLI.CTL0_L/H[DONE] bit in the LLI for that block. If the polled LLI.CTL0_L/H[DONE] bit is asserted, then this block transfer has completed. This LLI.CTL0_L/H[DONE] bit was cleared at the start of the transfer.

17. The end-of-block interrupt, int_block, is generated after the write-back of the control and status registers has completed.
18. The SATA DMA does not wait for the block interrupt to be cleared, but continues and fetches the next LLI from the memory location pointed to by the current LLP0 register and automatically reprograms the SAR0, CTL0_L/H, and LLP0 channel registers. The DAR0 register is left unchanged. The DMA transfer continues until the SATA DMA samples that the CTL0_L/H and LLP0 registers at the end of a block transfer match those described in Row 1 or Row 5 of *Table 21-2* (as discussed earlier). The SATA DMA then knows that the previously transferred block was the last block in the DMA transfer.

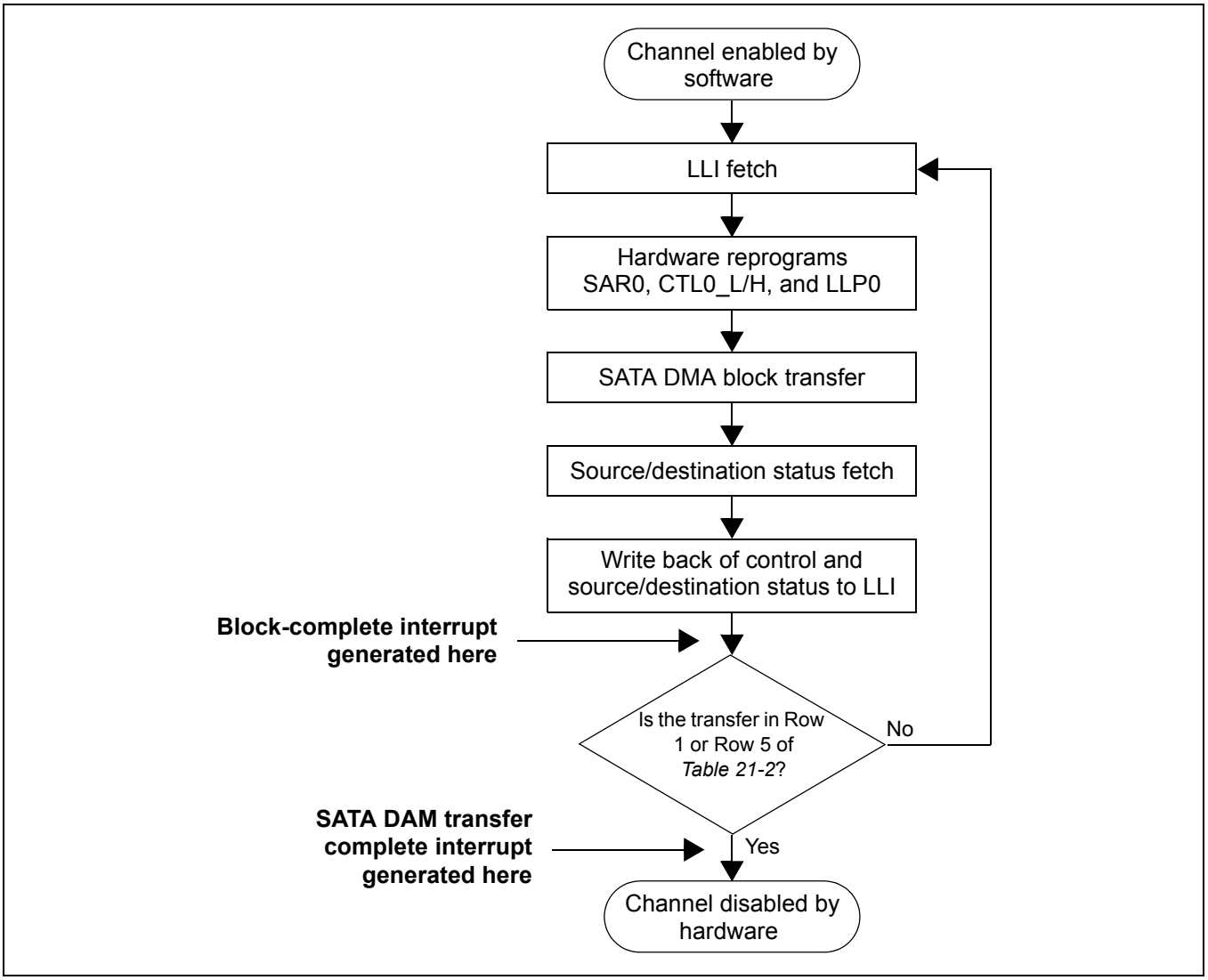
The SATA DMA transfer might look like that shown in the following figure. Note that the destination address is decrementing.

Figure 21-53. MBT with Linked List Source Address and Contiguous Destination Address



The DMA transfer flow is shown in the following figure.

Figure 21-54. MBT Flow for Source Address Auto-Reloaded and Contiguous Destination Address



21.13.6 Disabling a Channel Prior to Transfer Completion

Under normal operation, software enables a channel by writing 1 to the channel enable bit, AHBDMA0_ChEnReg[CH_EN], and hardware disables a channel on transfer completion by clearing the AHBDMA0_ChEnReg[CH_EN] bit.

The recommended way for software to disable a channel without losing data is to use the CH_SUSP bit in conjunction with the FIFO_EMPTY bit in the Channel Configuration Register (AHBDMA0_CFG0_L).

1. If software wishes to disable a channel prior to the DMA transfer completion, then it can set the AHBDMA0_CFG0_L[CH_SUSP] bit to tell the SATA DMA to halt all transfers from the source peripheral. Therefore, the channel FIFO receives no new data.
2. Software can now poll the AHBDMA0_CFG0_L[FIFO_EMPTY] bit until it indicates that the channel FIFO is empty.

3. The AHBDMA0_ChEnReg[CH_EN] bit can then be cleared by software once the channel FIFO is empty.

When AHBDMA0_CTL0_L[SRC_TR_WIDTH] < AHBDMA0_CTL0_L[DST_TR_WIDTH] and the AHBDMA0_CFG0_L[CH_SUSP] bit is high, AHBDMA0_CFG0_L[FIFO_EMPTY] is asserted once the contents of the FIFO do not permit a single word of AHBDMA0_CTL0_L[DST_TR_WIDTH] to be formed. However, there can still be data in the channel FIFO, but not enough to form a single transfer of AHBDMA0_CTL0_L[DST_TR_WIDTH]. In this scenario, once the channel is disabled, the remaining data in the channel FIFO is not transferred to the destination peripheral.

It is permissible to remove the channel from the suspension state by writing 0 to the AHBDMA0_CFG0_L.CH_SUSP register. The DMA transfer completes in the normal manner.

Note: If a channel is disabled by software, an active single or burst transaction is not guaranteed to receive an acknowledgement.

21.13.7 Abnormal Transfer Termination

A SATA DMA transfer may be terminated abruptly by software by clearing the channel enable bit, AHBDMA0_ChEnReg[CH_EN]. You must not assume that the channel is disabled immediately after the ChEnReg[CH_EN] is cleared. The CH_EN bit is cleared over the AHB slave interface. Consider this as a request to disable the channel. You must poll AHBDMA0_ChEnReg[CH_EN] and confirm that the channel is disabled by reading back 0. A case where the channel is not disabled after a channel disable request is where either the source or destination has received a split or retry response. The SATA DMA must keep attempting the transfer to the system HADDR that originally received the split or retry response until an OKAY response is returned. Otherwise, it is an AMBA protocol violation.

Software may terminate all channels abruptly by clearing the global enable bit in the SATA DMA Configuration Register (AHBDMA0_DmaCfgReg[31]). Again, you must not assume that all channels are disabled immediately after the AHBDMA0_DmaCfgReg[31] is cleared over the AHB slave interface. Consider this as a request to disable all channels. You must poll AHBDMA0_ChEnReg and confirm that all channels are disabled by reading back 0.

Note: If the channel enable bit is cleared while there is data in the channel FIFO, this data is not sent to the destination peripheral and is not present when the channel is re-enabled. For read-sensitive source peripherals, such as a source FIFO, this data is therefore lost. When the source is not a read-sensitive device (such as memory), disabling a channel without waiting for the channel FIFO to empty may be acceptable, since the data is available from the source peripheral upon request and is not lost.

21.14 SATA DMA Register Summary

The following table provides a summary of all the SATA DMA AHBDMA0_ registers along with a cross-reference to the location of the register details.

The starting memory location for these registers is 0x4 BFFD 0800.

Unless otherwise noted, all registers have a reset value of 0x0.

Table 21-3. SATA DMA Register Summary

Mnemonic	Description	Address	R/W	Page
Channel 0 Registers				
AHBDMA0_SAR0	Source Address Register 0	0x4_BFFD_0800	R/W	719
AHBDMA0_DAR0	Destination Address Register 0	0x4_BFFD_0808	R/W	719
AHBDMA0_LLP0	Linked List Pointer 0	0x4_BFFD_0810	R/W	720
AHBDMA0_CTL0_L	Control 0 Low	0x4_BFFD_0818	R/W	721
AHBDMA0_CTL0_H	Control 0 High	0x4_BFFD_081C	R/W	724
AHBDMA0_CFG0_L	Configuration 0 Low	0x4_BFFD_0840	R/W	725
AHBDMA0_CFG0_H	Configuration 0 High	0x4_BFFD_0844	R/W	727
AHBDMA0_SGR0	Source Gather Register 0	0x4_BFFD_0848	R/W	728
AHBDMA0_DSR0	Destination Scatter Register 0	0x4_BFFD_0850	R/W	728
Interrupt Registers				
AHBDMA0_RawTfr	Raw Status for IntTfr	0x4_BFFD_0AC0	R	730
AHBDMA0_RawBlock	Raw Status for IntBlock	0x4_BFFD_0AC8	R	730
AHBDMA0_RawSrcTran	Raw Status for IntSrcTran	0x4_BFFD_0AD0	R	730
AHBDMA0_RawDstTran	Raw Status for IntDstTran	0x4_BFFD_0AD8	R	730
AHBDMA0_RawErr	Raw Status for IntErr	0x4_BFFD_0AE0	R	730
AHBDMA0_StatusTfr	Status for IntTfr	0x4_BFFD_0AE8	R	730
AHBDMA0_StatusBlock	Status for IntBlock	0x4_BFFD_0AF0	R	730
AHBDMA0_StatusSrcTran	Status for IntSrcTran	0x4_BFFD_0AF8	R	730
AHBDMA0_StatusDstTran	Status for IntDstTran	0x4_BFFD_0B00	R	730
AHBDMA0_StatusErr	Status for IntErr	0x4_BFFD_0B08	R	730
AHBDMA0_MaskTfr	Mask for IntTfr	0x4_BFFD_0B10	R/W	730
AHBDMA0_MaskBlock	Mask for IntBlock	0x4_BFFD_0B18	R/W	730
AHBDMA0_MaskSrcTran	Mask for IntSrcTran	0x4_BFFD_0B20	R/W	730
AHBDMA0_MaskDstTran	Mask for IntDstTran	0x4_BFFD_0B28	R/W	730
AHBDMA0_MaskErr	Mask for IntErr	0x4_BFFD_0B30	R/W	730
AHBDMA0_ClearTfr	Clear for IntTfr	0x4_BFFD_0B38	W	731

Table 21-3. SATA DMA Register Summary (Continued)

Mnemonic	Description	Address	R/W	Page
AHBDMA0_ClearBlock	Clear for IntBlock	0x4_BFFD_0B40	W	731
AHBDMA0_ClearSrcTran	Clear for IntSrcTran	0x4_BFFD_0B48	W	731
AHBDMA0_ClearDstTran	Clear for IntDstTran	0x4_BFFD_0B50	W	731
AHBDMA0_ClearErr	Clear for IntErr	0x4_BFFD_0B58	W	731
AHBDMA0_StatusInt	Status for each interrupt type	0x4_BFFD_0B60	W	732
Software Handshaking Registers				
AHBDMA0_RegSrcReg	Source Software Transaction Request Register	0x4_BFFD_0B68	R/W	732
AHBDMA0_RegDstReg	Destination Software Transaction Request Register	0x4_BFFD_0B70	R/W	733
AHBDMA0_SglRegSrcReg	Single Source Transaction Request Register	0x4_BFFD_0B78	R/W	733
AHBDMA0_SglRegDstReg	Single Destination Transaction Request Register	0x4_BFFD_0B80	R/W	734
AHBDMA0_LstSrcReg	Last Source Transaction Request Register	0x4_BFFD_0B88	R/W	734
AHBDMA0_LstDstReg	Last Destination Transaction Request Register	0x4_BFFD_0B90	R/W	735
Miscellaneous Registers				
AHBDMA0_DmaCfgReg	DMA Configuration Register	0x4_BFFD_0B98	R/W	735
AHBDMA0_ChEnReg	Channel Enable Register	0x4_BFFD_0BA0	R/W	736
AHBDMA0_DmaIdReg	DMA ID Register	0x4_BFFD_0BA8	R	736
AHBDMA0_DmaTestReg	DMA Test Register	0x4_BFFD_0BB4	R/W	737
AHBDMA0_DMA_COMP_PARAMS_2_L	DMA Component Parameters Register 2 Low	0x4_BFFD_0BE8	R	737
AHBDMA0_DMA_COMP_PARAMS_2_H	DMA Component Parameters Register 2 High	0x4_BFFD_0BEC	R	738
AHBDMA0_DMA_COMP_PARAMS_1_L	DMA Component Parameters Register 1 Low	0x4_BFFD_0BF0	R	739
AHBDMA0_DMA_COMP_PARAMS_1_H	DMA Component Parameters Register 1 High	0x4_BFFD_0BF4	R	739
AHBDMA0_DMA_COMP_TYPE	DMA Component Type	0x4_BFFD_0BF8	R	740
AHBDMA0_DMA_COMP_VER	DMA Component Version	0x4_BFFD_0BFC	R	740

21.15 SATA DMA Register Descriptions

The following sections contain the descriptions of the individual registers.

21.15.1 Channel Registers

The channel 0 registers consist of the following

- AHBDMA0_CFG0_L/H – Configuration
- AHBDMA0_CTL0_L/H – Control
- AHBDMA0_DAR0 – Destination address register
- AHBDMA0_DSR0 – Destination scatter register
- AHBDMA0_LLP0 – Linked list pointer

- AHBDMA0_SAR0 – Source address register
- AHBDMA0_SGR0 – Source gather register

The AHBDMA0_SAR0, AHBDMA0_DAR0, AHBDMA0_LLP0, AHBDMA0_CTL0_L/H, and AHBDMA0_CFG0_L/H channel registers should be programmed prior to enabling the channel. However, if an LLI update occurs before commencing data transfer, AHBDMA0_SAR0 and AHBDMA0_DAR0 may not need to be programmed prior to enabling the channel; refer to Rows 6 to 10 in *Table 21-2* on page 693. It is an illegal register access when a write to the AHBDMA0_SAR0, AHBDMA0_DAR0, AHBDMA0_LLP0, AHBDMA0_CTL0_L/H, AHBDMA0_SGR0, or AHBDMA0_DSR0 registers occurs when the channel is enabled.

21.15.1.1 Source Address Register for Channel 0 (AHBDMA0_SAR0)

The starting source address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the source address of the current AHB transfer.

Note: Program the SAR address to be aligned to AHBDMA0_CTL0_L[SRC_TR_WIDTH].

For information on how the AHBDMA0_SAR0 is updated at the start of each DMA block for multiblock transfers, refer to *Table 21-2* on page 693.

Figure 21-55. Source Address Register for Channel 0 (AHBDMA0_SAR0)

Bits	Name	R/W	Description
0:31	AHBDMA0_SAR	R/W	Current Source Address of DMA transfer Updated after each source transfer. The SINC field in the AHBDMA0_ CTL0_L register determines whether the address increments, decrements, or is left unchanged on every source transfer throughout the block transfer.

21.15.1.2 Destination Address Register for Channel 0 (AHBDMA0_DAR0)

The starting destination address is programmed by software before the DMA channel is enabled, or by an LLI update before the start of the DMA transfer. While the DMA transfer is in progress, this register is updated to reflect the destination address of the current AHB transfer.

Note: Program the DAR to be aligned to AHBDMA0_CTL0_L[DST_TR_WIDTH].

For information on how the AHBDMA0_DAR0 is updated at the start of each DMA block for multiblock transfers, refer to *Table 21-2* on page 693 and *Programming Examples* on page 697.

Figure 21-56. Destination Address Register for Channel 0 (AHBDMA0_DAR0)

Bits	Name	R/W	Description
0:31	DAR	R/W	Current Destination address of DMA transfer Updated after each destination transfer. The DINC field in the AHBDMA0_CTL0_L register determines whether the address increments, decrements, or is left unchanged on every destination transfer throughout the block transfer.

21.15.1.3 Hardware Realignment of SAR/DAR Registers

In a particular circumstance, during contiguous multiblock DMA transfers, the destination address can become misaligned between the end of one block and the start of the next block. When this situation occurs, SATA DMA re-aligns the destination address before the start of the next block.

Consider the following example. If the block length is 9, the source transfer width is 16 (halfword), and the destination transfer width is 32 (word)—the destination is programmed for contiguous block transfers—then the destination performs four word transfers followed by a halfword transfer to complete the block transfer to the destination. At the end of the destination block transfer, the address is aligned to a 16-bit transfer as the last AMBA transfer is halfword. This is misaligned to the programmed transfer size of 32 bits for the destination. However, for contiguous destination multiblock transfers, SATA DMA re-aligns the DAR address to the nearest 32-bit address (next 32-bit address upwards if address control is incrementing or next address downwards if address control is decrementing).

The destination address is automatically realigned by the SATA DMA in the following DMA transfer setup scenario:

- Contiguous multiblock transfers on destination side, AND
- DST_TR_WIDTH > SRC_TR_WIDTH, AND
- (BLOCK_TS * SRC_TR_WIDTH)/DST_TR_WIDTH != integer (where SRC_TR_WIDTH, DST_TR_WIDTH is byte width of transfer)

21.15.1.4 Linked List Pointer Register for Channel 0 (AHBDMA0_LLP0)

Note: Program this register to point to the first Linked List Item (LLI) in memory prior to enabling the channel if block chaining is enabled – Rows 6 to 10 of *Table 21-2* on page 693.

The AHBDMA0_ LLP0 register has two functions:

- The logical result of the equation AHBDMA0_LLP0[LOC] = 0 is used to set up the type of DMA transfer—single or multiblock. *Table 21-2* on page 693 shows how the method of updating the channel registers is a function of AHBDMA0_LLP0[LOC] != 0. If AHBDMA0_LLP0[LOC] is set to 0x0, then transfers using linked lists are *not* enabled. This register must be programmed prior to enabling the channel in order to set up the transfer type.
- AHBDMA0_LLP0[LOC] = 0 contains the pointer to the next LLI for block chaining using linked lists. The AHBDMA0_LLP0 register can also point to the address where write back of the control and source/destination status information occur after block completion.

Figure 21-57. Linked List Pointer Register for Channel 0 (AHBDMA0_LLP0)

Bits	Name	R/W	Description
0:29	LOC	R/W	Starting Address In Memory of next LLI if block chaining is enabled Note that the two LSBs of the starting address are not stored because the address is assumed to be aligned to a 32-bit boundary. LLI accesses are always 32-bit accesses (Hsize = 2) and cannot be changed or programmed to anything other than 32-bit.
30:31	LMS	R/W	List Master Select Identifies the AHB layer/interface where the memory device that stores the next linked list item resides. 00 = AHB master 1 01 = AHB master 2 1x = Reserved The maximum value of this field that can be read back is 1.

21.15.1.5 Control Register for Channel 0 Low (AHBDMA0_CTL0_L)

This register contains fields that control the DMA transfer.

The AHBDMA0_CTL0_L register is part of the block descriptor when block chaining is enabled. It can be varied on a block-by-block basis within a DMA transfer when block chaining is enabled. For information about the behavior of this register between blocks, refer to *Multiblock Transfers (MBTs)* on page 692.

Note: Program this register prior to enabling the channel.

Figure 21-58. Control Register for Channel 0 Low (AHBDMA0_CTL0_L)

Bits	Name	R/W	Description
0:2		N/A	Reserved
3	LLP_SRC_EN	R/W	Block chaining is enabled on the source side only if the LLP_SRC_EN field is high and AHBDMA0_LLP0.LOC is non-zero. **Reset Value:** 0x0
4	LLP_DST_EN	R/W	Block chaining is enabled on the destination side only if the LLP_DST_EN field is high and LLP*x*.LOC is non-zero. **Reset Value:** 0x0
5:6	SMS	R/W	Source Master Select Identifies the Master Interface layer from which the source device (peripheral or memory) is accessed. 00 = AHB master 1 01 = AHB master 2 1x = Reserved The maximum value of this field that can be read back is 1. **Reset Value:** 0x0
7:8	DMS	R/W	Destination Master Select Identifies the Master Interface layer where the destination device (peripheral or memory) resides. 00 = AHB master 1 01 = AHB master 2 1x = Reserved The maximum value of this field that can be read back is 1. **Reset Value:** 0x0
9:11	TT_FC	R/W	Transfer Type and Flow Control The following transfer types are supported: • Memory to Memory • Memory to Peripheral • Peripheral to Memory • Peripheral to Peripheral Flow Control can be assigned to the SATA DMA, the source peripheral, or the destination peripheral. **Reset Value:** 0x0 For multiblock transfers using linked list operation, TT_FC must be constant for all blocks of this multiblock transfer.
12		N/A	Reserved

Figure 21-58. Control Register for Channel 0 Low (AHBDMA0_CTL0_L) (Continued)

Bits	Name	R/W	Description
13	DST_SCATTER_EN	R/W	Destination scatter enable bit: 0 = Scatter disabled 1 = Scatter enabled Scatter on the destination side is applicable only when the AHBDMA0_CTL0_L.DINC bit indicates an incrementing or decrementing address control. **Reset Value:** 0x0
14	SRC_GATHER_EN	R/W	Source gather enable bit: 0 = Gather disabled 1 = Gather enabled Gather on the source side is applicable only when the AHBDMA0_CTL0_L.SINC bit indicates an incrementing or decrementing address control. **Reset Value:** 0x0
15:17	SRC_MSIZE	R/W	Source Burst Transaction Length Number of data items, each of width AHBDMA0_CTL0_L[SRC_TR_WIDTH], to be read from the source every time a source burst transaction request is made from either the corresponding hardware or software handshaking interface. **Note:** This value is not related to the AHB bus master HBURST bus. **Reset Value:** 0x1 The Maximum value for this field is 0x2, which corresponds to a size of 32.
18:20	DEST_MSIZE	R/W	Destination Burst Transaction Length Number of data items, each of width AHBDMA0_CTL0_L[DST_TR_WIDTH], to be written to the destination every time a destination burst transaction request is made from either the corresponding hardware or software handshaking interface. **Note:** This value is not related to the AHB bus master HBURST bus. **Reset Value:** 0x1 The Maximum value for this field is 0x2, which corresponds to a size of 32.
21:22	SINC	R/W	Source Address Increment Indicates whether to increment or decrement the source address on every source transfer. If the device is fetching data from a source peripheral FIFO with a fixed address, then set this field to "No change." 00 = Increment 01 = Decrement 1x = No change **Note:** Incrementing or decrementing is done for alignment to the next AHBDMA0_CTL0_L[SRC_TR_WIDTH] boundary. **Reset Value:** 0x0
23:24	DINC	R/W	Destination Address Increment Indicates whether to increment or decrement the destination address on every destination transfer. If your device is writing data to a destination peripheral FIFO with a fixed address, then set this field to "No change." 00 = Increment 01 = Decrement 1x = No change **Note:** Incrementing or decrementing is done for alignment to the next AHBDMA0_CTL0_L[DST_TR_WIDTH] boundary. **Reset Value:** 0x0
25:27	SRC_TR_WIDTH	R/W	Source Transfer Width Mapped to AHB bus "hsize." For a non-memory peripheral, typically the peripheral (source) FIFO width. This field is hard coded to 0x2, which corresponds to a width of 32.

Figure 21-58. Control Register for Channel 0 Low (AHBDMA0_CTL0_L) (Continued)			
Bits	**Name**	**R/W**	**Description**
28:30	DST_TR_WIDTH	R/W	Destination Transfer Width Mapped to AHB bus "hsize." For a non-memory peripheral, typically rgw peripheral (destination) FIFO width. This value must be less than or equal to This field is hard coded to 0x2, which corresponds to a width of 32.
31	INT_EN	R/W	Interrupt Enable Bit If set, then all interrupt generating sources are enabled. **Reset Value:** 0x1

Table 21-4. AHBDMA0_CTL0_L.SRC_MSIZE and DEST_MSIZE Decoding

AHBDMA0_CTL0_L.SRC_MSIZE/AHBDMA0_CTL0_L.DEST_MSIZE	Number of data items to be transferred (of width AHBDMA0_CTL0_L.SRC_TR_WIDTH or AHBDMA0_CTL0_L.DST_TR_WIDTH)
000	1
001	4
010	8
011	16
100	32
101	64
110	128
111	256

Table 21-5. AHBDMA0_CTL0_L.SRC_TR_WIDTH and AHBDMA0_CTL0_L.DST_TR_WIDTH Decoding

AHBDMA0_CTL0_L.SRC_TR_WIDTH/AHBDMA0_CTL0_L.DST_TR_WIDTH	Size (bits)
000	8
001	16
010	32
011	64
100	128
101	256
11x	256

Table 21-6. AHBDMA0_CTL0_L.TT_FC Field Decoding

HBDMA0_CTL0_L.TT_FC Field	Transfer Type	Flow Controller
000	Memory to Memory	SATA DMA
001	Memory to Peripheral	SATA DMA
010	Peripheral to Memory	SATA DMA
011	Peripheral to Peripheral	SATA DMA
100	Peripheral to Memory	Peripheral
101	Peripheral to Peripheral	Source Peripheral
110	Memory to Peripheral	Peripheral
111	Peripheral to Peripheral	Destination Peripheral

21.15.1.6 Control Register for Channel 0 High (AHBDMA0_CTL0_H)

This register contains fields that control the DMA transfer.

The AHBDMA0_CTL0_H register is part of the block descriptor when block chaining is enabled. It can be varied on a block-by-block basis within a DMA transfer when block chaining is enabled. For information about the behavior of this register between blocks, refer to *Multiblock Transfers (MBTs)* on page 692.

If status write-back is enabled, this register is written to the control register location of the LLI (linked list item) in system memory at the end of the block transfer.

Note: Program this register prior to enabling the channel.

Figure 21-59. Control Register for Channel 0 High (AHBDMA0_CTL0_H)

Bits	Name	R/W	Description
0:18		N/A	Reserved
19	DONE	R/W	Done bit If status write back is enabled, this register is written to the control register location of the Linked List Item (LLI) in system memory at the end of the block transfer with the DONE bit set. Software can poll the LLI AHBDMA0_CTL0_H.DONE bit to see when a block transfer is complete. The LLI AHBDMA0_CTL0_H.DONE bit should be cleared when the linked lists are set up in memory prior to enabling the channel. LLI accesses are always 32-bit accesses (Hsize = 2) and cannot be changed or programmed to anything other than 32-bit. For more information, refer to *Multiblock Transfers (MBTs)* on page 692. **Reset Value:** 0x0

Figure 21-59. Control Register for Channel 0 High (AHBDMA0_CTL0_H) (Continued)			
Bits	**Name**	**R/W**	**Description**
20:31	BLOCK_TS	R/W	Block Transfer Size When the SATA DMA is the flow controller, the user writes this field before the channel is enabled in order to indicate the block size. The number programmed into BLOCK_TS indicates the total number of single transactions to perform for every block transfer; a single transaction is mapped to a single AMBA beat. **Width:** The width of the single transaction is determined by AHBDMA0_CTL0_L.SRC_TR_WIDTH Once the transfer starts, the read back value is the total number of data items already read from the source peripheral, regardless of what is the flow controller. When the source or destination peripheral is assigned as the flow controller, then the maximum block size that can be read back saturates at 0xFFF, but the actual block size can be greater. Refer to *Table 21-2* on page 693. **Reset Value:** 0x2

21.15.1.7 Configuration Register for Channel 0 Low (AHBDMA0_CFG0_L)

This register contains fields that configure the DMA transfer. The channel configuration register remains fixed for all blocks of a multiblock transfer.

Note: Program this register prior to enabling the channel.

Figure 21-60. Configuration Register for Channel 0 Low (AHBDMA0_CFG0_L)			
Bits	**Name**	**R/W**	**Description**
0	RELOAD_DST	R/W	Automatic Destination Reload The AHBDMA0_DAR0 register can be automatically reloaded from its initial value at the end of every block for multiblock transfers. A new block transfer is then initiated. This register does not exist if the configuration parameter DMAH_CH*x*_MULTI_BLK_EN is not selected; in this case, the read back value is always 0.
1	RELOAD_SRC	R/W	Automatic Source Reload The AHBDMA0_SAR0 register can be automatically reloaded from its initial value at the end of every block for multiblock transfers. A new block transfer is then initiated. For conditions under which this occurs, refer to *Table 21-2* on page 693.
2:11	MAX_ABRST	R/W	Maximum AMBA Burst Length The read back value is always 0, the maximum AMBA burst length cannot be limited by software.
12	SRC_HS_POL	R/W	Source Handshaking Interface Polarity 0 = Active high 1 = Active low
13	DST_HS_POL	R/W	Destination Handshaking Interface Polarity 0 = Active high 1 = Active low
14	LOCK_B	R/W	Bus Lock Bit The read back value is always 0.
15	LOCK_CH	R/W	The read back value is always 0.

Figure 21-60. Configuration Register for Channel 0 Low (AHBDMA0_CFG0_L) (Continued)

Bits	Name	R/W	Description
16:17	LOCK_B_L	R/W	The read back value is always 0.
18:19	LOCK_CH_L	R/W	The read back value is always 0.
20	HS_SEL_SRC	R/W	Source Software or Hardware Handshaking Select This register selects which of the handshaking interfaces – hardware or software – is active for source requests on this channel. 0 = Hardware handshaking interface. Software-initiated transaction requests are ignored. 1 = Software handshaking interface. Hardware-initiated transaction requests are ignored. If the source peripheral is memory, then this bit is ignored. **Reset Value:** 1
21	HS_SEL_DST	R/W	Destination Software or Hardware Handshaking Select This register selects which of the handshaking interfaces – hardware or software – is active for destination requests on this channel. 0 = Hardware handshaking interface. Software-initiated transaction requests are ignored. 1 = Software handshaking interface. Hardware- initiated transaction requests are ignored. If the destination peripheral is memory, then this bit is ignored. **Reset Value:** 1
22	FIFO_EMPTY	R	Indicates if there is data left in the channel FIFO. Can be used in conjunction with AHBDMA0_CFG0_L.CH_SUSP to cleanly disable a channel. For more information, refer to *Disabling a Channel Prior to Transfer Completion* on page 715. 1 = Channel FIFO empty 0 = Channel FIFO not empty
23	CH_SUSP	R/W	Channel Suspend Suspends all DMA data transfers from the source until this bit is cleared. There is no guarantee that the current transaction will complete. Can also be used in conjunction with AHBDMA0_CFG0_L[FIFO_EMPTY] to cleanly disable a channel without losing any data. 0 = Not suspended. 1 = Suspend DMA transfer from the source. For more information, refer to *Disabling a Channel Prior to Transfer Completion* on page 715.
24:26	CH_PRIOR	R/W	**Channel priority** Always set to 0x0 since only one channel. **Reset Value:** Channel number For example: Chan0 = 0 Chan1 = 1
27:31		N/A	Reserved

21.15.1.8 Configuration Register for Channel 0 High (AHBDMA0_CFG0_H)

This register contains fields that configure the DMA transfer. The channel configuration register remains fixed for all blocks of a multiblock transfer.

Note: Program this register prior to enabling the channel.

Figure 21-61. Configuration Register for Channel 0 High (AHBDMA0_CFG0_H)

Bits	Name	R/W	Description
0:16		N/A	Reserved
17:20	DEST_PER	R/W	Assigns the hardware handshaking interface (0) to the destination of channel 0 if the AHBDMA0_CFG0_L[HS_SEL_DST] field is 0; otherwise, this field is ignored. The channel can then communicate with the destination peripheral connected to that interface through the assigned hardware handshaking interface. **Note:** For correct DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface. Sine there is only one handshaking interface (0) only one valid value is 0x0.
21:24	SRC_PER	R/W	Assigns the hardware handshaking interface (0) to the source of channel 0 if the AHBDMA0_CFG0_L[HS_SEL_SRC] field is 0; otherwise, this field is ignored. The channel can then communicate with the source peripheral connected to that interface through the assigned hardware handshaking interface. **Note:** For correct SATA DMA operation, only one peripheral (source or destination) should be assigned to the same handshaking interface. Since there is only one handshaking interface (0) only one valid value is 0x0.
25	SS_UPD_EN	R/W	The read back value is always 0.
26	DS_UPD_EN	R/W	The read back value is always 0.
27:29	PROTCTL	R/W	Protection Control bits used to drive the AHB HPROT[3:1] bus The *AMBA Specification* recommends that the default value of HPROT indicates a non-cached, non-buffered, privileged data access. The reset value is used to indicate such an access. HPROT[0] is tied high because all transfers are data accesses, as there are no opcode fetches. There is a one-to-one mapping of these register bits to the HPROT[3:1] master interface signals. **Reset Value:** 1
30	FIFO_MODE	R/W	FIFO Mode Select Determines how much space or data needs to be available in the FIFO before a burst transaction request is serviced. 0 = Space/data available for single AHB transfer of the specified transfer width. 1 = Space/data available is greater than or equal to half the FIFO depth for destination transfers and less than half the FIFO depth for source transfers. The exceptions are at the end of a burst transaction request or at the end of a block transfer.
31	FCMODE	R/W	Flow Control Mode Determines when source transaction requests are serviced when the Destination Peripheral is the flow controller. 0 = Source transaction requests are serviced when they occur. Data pre-fetching is enabled. 1 = Source transaction requests are not serviced until a destination transaction request occurs. In this mode, the amount of data transferred from the source is limited so that it is guaranteed to be transferred to the destination prior to block termination by the destination. Data pre-fetching is disabled.

21.15.1.9 Source Gather Register for Channel 0 (AHBDMA0_SGR0)

The Source Gather register contains two fields:

- Source gather count field (AHBDMA0_SGR0[SGC]) – Specifies the number of contiguous source transfers of AHBDMA0_CTL0_L[SRC_TR_WIDTH] between successive gather intervals. This is defined as a gather boundary.
- Source gather interval field (AHBDMA0_SGR0[SGI]) – Specifies the source address increment/decrement in multiples of AHBDMA0_CTL0_L[SRC_TR_WIDTH] on a gather boundary when gather mode is enabled for the source transfer.

The AHBDMA0_CTL0_L[SINC] field controls whether the address increments or decrements. When the AHBDMA0_CTL0_L[SINC] field indicates a fixed-address control, then the address remains constant throughout the transfer and the AHBDMA0_SGR0 register is ignored.

Figure 21-62. Source Gather Register for Channel 0 (AHBDMA0_SGR0)

Bits	Name	R/W	Description
0:11	SGC	R/W	Source gather count. Source contiguous transfer count between successive gather boundaries.
12:31	SGI	R/W	Source gather interval.

21.15.1.10 Destination Scatter Register for Channel 0 (AHBDMA0_DSR0)

The Destination Scatter register contains two fields:

- Destination scatter count field (AHBDMA0_DSR0[DSC]) – Specifies the number of contiguous destination transfers of AHBDMA0_CTL0_L[DST_TR_WIDTH] between successive scatter boundaries.
- Destination scatter interval field (AHBDMA0_DSR0[DSI]) – Specifies the destination address increment/decrement in multiples of AHBDMA0_CTL0_L[DST_TR_WIDTH] on a scatter boundary when scatter mode is enabled for the destination transfer.

The AHBDMA0_CTL0_L[DINC] field controls whether the address increments or decrements. When the AHBDMA0_CTL0_L[DINC] field indicates a fixed address control, then the address remains constant throughout the transfer and the AHBDMA0_DSR0 register is ignored.

Figure 21-63. Destination Scatter Register for Channel 0 (AHBDMA0_DSR0)

Bits	Name	R/W	Reset	Description
0:11	DSC	R/W	0x0	Destination scatter count. Destination contiguous transfer count between successive scatter boundaries.
12:31	DSI	R/W	0x0	Destination scatter interval.

21.15.2 Interrupt Registers

The following sections describe the registers pertaining to interrupts, their status, and how to clear them. For each channel, there are five types of interrupt sources:

- AHBDMA0_IntBlock – Block Transfer Complete Interrupt

This interrupt is generated on DMA block transfer completion to the destination peripheral.

- AHBDMA0_IntDstTran – Destination Transaction Complete Interrupt

  This interrupt is generated after completion of the last AHB transfer of the requested single/burst transaction from the handshaking interface (either the hardware or software handshaking interface) on the destination side.

  Note: If the destination for a channel is memory, then that channel will never generate the IntDstTran interrupt. Because of this, the corresponding bit in this field will not be set.

- AHBDMA0_IntErr – Error Interrupt

  This interrupt is generated when an ERROR response is received from an AHB slave on the HRESP bus during a DMA transfer. In addition, the DMA transfer is cancelled and the channel is disabled.

- AHBDMA0_IntSrcTran – Source Transaction Complete Interrupt

  This interrupt is generated after completion of the last AHB transfer of the requested single/burst transaction from the handshaking interface (either the hardware or software handshaking interface) on the source side.

  Note: If the source or destination is memory, then AHBDMA0_IntSrcTran/AHBDMA0_IntDstTran interrupts should be ignored, as there is no concept of a "DMA transaction level" for memory.

- AHBDMA0_IntTfr – DMA Transfer Complete Interrupt

  This interrupt is generated on DMA transfer completion to the destination peripheral.

There are several groups of interrupt related registers:

- AHBDMA0_RawBlock, AHBDMA0_RawDstTran, AHBDMA0_RawErr, AHBDMA0_RawSrcTran, AHBDMA0_RawTfr
- AHBDMA0_StatusBlock, AHBDMA0_StatusDstTran, AHBDMA0_StatusErr, AHBDMA0_StatusSrcTran, AHBDMA0_StatusTfr
- AHBDMA0_MaskBlock, AHBDMA0_MaskDstTran, AHBDMA0_MaskErr, AHBDMA0_MaskSrcTran, AHBDMA0_MaskTfr
- AHBDMA0_ClearBlock, AHBDMA0_ClearDstTran, AHBDMA0_ClearErr, AHBDMA0_ClearSrcTran, AHBDMA0_ClearTfr
- AHBDMA0_StatusInt

When a channel has been enabled to generate interrupts, the following is true:

- Interrupt events are stored in the Raw Status registers.
- The contents of the Raw Status registers are masked with the contents of the Mask registers.
- The masked interrupts are stored in the Status registers.
- The contents of the Status registers are used to drive the int_* port signals.
- Writing to the appropriate bit in the Clear registers clears an interrupt in the Raw Status registers and the Status registers on the same clock cycle.

The contents of each of the five Status registers is ORed to produce a single bit for each interrupt type in the Combined Status register; that is, AHBDMA0_StatusInt.

Note: The AHBDMA0_ CTL0_L[INT_EN] bit must be set for an enabled channel to generate any interrupts.

21.15.2.1 Interrupt Raw Status Registers (Raw Block, DstTran, Err, SrcTran, Tfr)

Interrupt events are stored in these Raw Interrupt Status registers before masking: AHBDMA0_RawBlock, AHBDMA0_RawDstTran, AHBDMA0_RawErr, AHBDMA0_RawSrcTran, and AHBDMA0_RawTfr.

Each bit in these registers is cleared by writing a 1 to the corresponding location in the AHBDMA0_ClearTfr, AHBDMA0_ClearBlock, AHBDMA0_ClearSrcTran, AHBDMA0_ClearDstTran, AHBDMA0_ClearErr registers.

Note: Write access is available to these registers for software testing purposes only. Under normal operation, writes to these registers are not recommended.

Figure 21-64. Interrupt Raw Status Registers (Raw Block, DstTran, Err, SrcTran, Tfr)

Bits	Name	R/W	Description
0:30		N/A	Reserved
31	RAW	R/W	Raw interrupt status.

21.15.2.2 Interrupt Status Registers (Status Block, Tran, Err, SrcTran, Tfr)

All interrupt events from all channels are stored in these Interrupt Status registers after masking: AHBDMA0_StatusBlock, AHBDMA0_StatusDstTran, AHBDMA0_StatusErr, AHBDMA0_StatusSrcTran, and AHBDMA0_StatusTfr. The contents of these registers are used to generate the interrupt signals (int or int_n bus, depending on interrupt polarity) leaving the SATA DMA.

Figure 21-65. Interrupt Status Registers (Status Block, DstTran, Err, SrcTran, Tfr)

Bits	Name	R/W	Description
0:30		N/A	Reserved
31	STATUS	R	Interrupt status.

21.15.2.3 Interrupt Mask Registers (Mask Block, DstTran, Err, SrcTran, Tfr)

The contents of the Raw Status registers are masked with the contents of the Mask registers: AHBDMA0_MaskBlock, AHBDMA0_MaskDstTran, AHBDMA0_MaskErr, AHBDMA0_MaskSrcTran, and AHBDMA0_MaskTfr.

When the source peripheral of DMA channel 0 is memory, then the source transaction complete interrupt, AHBDMA0_MaskSrcTran[31], must be masked to prevent an erroneous triggering of an interrupt on the int_combined signal. Similarly, when the destination peripheral of DMA channel 0 is memory, then the destination transaction complete interrupt, AHBDMA0_MaskDstTran[31], must be masked to prevent an erroneous triggering of an interrupt on the int_combined(_n) signal.

A channel INT_MASK bit will be written only if the corresponding mask write enable bit in the INT_MASK_WE field is asserted on the same AHB write transfer. This allows software to set a mask bit without performing a read-modified write operation. For example, writing hex 01x1 to the AHBDMA0_MaskTfr register writes a 1 into AHBDMA0_MaskTfr[31], while AHBDMA0_MaskTfr[24:30] remains unchanged. Writing hex 00*xx* leaves AHBDMA0_MaskTfr[24:31] unchanged.

Writing a 1 to any bit in these registers unmasks the corresponding interrupt, thus allowing the SATA DMA to set the appropriate bit in the Status registers and int_* port signals.

Figure 21-66. Interrupt Mask Registers (Mask Block, DstTran, Err, SrcTran, Tfr)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	INT_MASK_WE	W	Interrupt Mask Write Enable 0 = Write disabled 1 = Write enabled **Reset Value:** 0x0
24:30		N/A	Reserved
31	INT_MASK	R/W	Interrupt Mask 0 = Masked 1 = Unmasked **Reset Value:** 0x0

21.15.2.4 Interrupt Clear Registers (Clear Block, DstTran, Err, SrcTran, Tfr)

Each bit in the Raw Status and Status registers is cleared on the same cycle by writing a 1 to the corresponding location in the Clear registers: AHBDMA0_ClearBlock, AHBDMA0_ClearDstTran, AHBDMA0_ClearErr, AHBDMA0_ClearSrcTran, and AHBDMA0_ClearTfr. Writing a 0 has no effect. These registers are not readable.

Figure 21-67. Interrupt Clear Registers (Clear Block, DstTran, Err, SrcTran, Tfr)

Bits	Name	R/W	Description
0:30		N/A	Reserved
31	CLEAR	W	Interrupt clear. 0 = No effect 1 = Clear interrupt

21.15.2.5 Combined Interrupt Status Register (AHBDMA0_StatusInt)

The contents of each of the five Status registers – AHBDMA0_StatusTfr, AHBDMA0_StatusBlock, AHBDMA0_StatusSrcTran, AHBDMA0_StatusDstTran, AHBDMA0_StatusErr – are ORed to produce a single bit for each interrupt type in the Combined Status register (AHBDMA0_StatusInt). This register is read only.

Figure 21-68. Combined Interrupt Status (AHBDMA0_StatusInt)

Bits	Name	R/W	Description
0:26		N/A	Reserved
27	ERR	R	OR of the contents of AHBDMA0_StatusErr register.
28	DSTT	R	OR of the contents of AHBDMA0_StatusDst register.
29	SRCT	R	OR of the contents of AHBDMA0_StatusSrcTran register.
30	BLOCK	R	OR of the contents of AHBDMA0_StatusBlock register.
31	TFR	R	OR of the contents of AHBDMA0_StatusTfr register.

21.15.3 Software Handshaking Registers

The registers that comprise the software handshaking registers allow software to initiate single or burst transaction requests in the same way that handshaking interface signals do in hardware.

Setting AHBDMA0_CFG0_L.HS_SEL_SRC to 1 enables software handshaking on the source of channel 0.
Setting AHBDMA0_CFG0_L.HS_SEL_DST to 1 enables software handshaking on the destination of channel 0.

21.15.3.1 Source Software Transaction Request Register (AHBDMA0_ReqSrcReg)

AHBDMA0_ReqSrcReg[31] is ignored when software handshaking is not enabled for the source of channel 0.

Figure 21-69. Source Software Transaction Request Register (AHBDMA0_ReqSrcReg)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	SRC_REQ_WE	W	Source request write enable 0 = Write disabled 1 = Write enabled
24:30	Undefined	N/A	Reserved
31	SRC_REQ	R/W	Source request

A channel SRC_REQ bit is written only if the corresponding channel write enable bit in the SRC_REQ_WE field is asserted on the same AHB write transfer. For example, writing hex 0101 writes a 1 into AHBDMA0_ReqSrcReg[31], while AHBDMA0_ReqSrcReg[24:30] remains unchanged. Writing hex 00*xx* leaves AHBDMA0_ReqSrcReg[24:31] unchanged. This allows software to set a bit in the AHBDMA0_ReqSrcReg register without performing a read-modified write operation.

The functionality of this register depends on whether the source is a flow control peripheral or not.

21.15.3.2 Destination Software Transaction Request Register (AHBDMA0_ReqDstReg)

AHBDMA0_ReqDstReg[31] is ignored when software handshaking is not enabled for the source of channel 0.

Figure 21-70. Destination Software Transaction Request Register (AHBDMA0_ReqDstReg)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	DST_REQ_WE	W	Destination request write enable 0 = Write disabled 1 = Write enabled
24:30		N/A	Reserved
31	DST_REQ	R/W	Destination request

A channel DST_REQ bit is written only if the corresponding channel write enable bit in the DST_REQ_WE field is asserted on the same AHB write transfer.

The functionality of this register depends on whether the destination is a flow control peripheral or not.

21.15.3.3 Single Source Transaction Request Register (AHBDMA0_SglReqSrcReg)

AHBDMA0_SglReqSrcReg[31] is ignored when software handshaking is not enabled for the source of channel 0.

Figure 21-71. Single Source Transaction Request Register (AHBDMA0_SglReqSrcReg)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	SRC_SGLREQ_WE	W	Single write enable 0 = Write disabled 1 = Write enabled
24:30		N/A	Reserved
31	SRC_SGLREQ	R/W	Source single request

A channel SRC_SGLREQ bit is written only if the corresponding channel write enable bit in the SRC_SGLREQ_WE field is asserted on the same AHB write transfer.

The functionality of this register depends on whether the source is a flow control peripheral or not. For a description of when the source is not a flow controller.

21.15.3.4 Single Destination Transaction Request Register (AHBDMA0_SglReqDstReg)

AHBDMA0_SglReqDstReg[31] is ignored when software handshaking is not enabled for the destination of channel 0.

Figure 21-72. Single Destination Transaction Request Register (AHBDMA0_SglReqDstReg)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	DST_SGLREQ_WE	W	Destination write enable 0 = Write disabled 1 = Write enabled
24:30		N/A	Reserved
31	DST_SGLREQ	R/W	Destination single or burst request

A channel DST_SGLREQ bit is written only if the corresponding channel write enable bit in the DST_SGLREQ_WE field is asserted on the same AHB write transfer.

The functionality of this register depends on whether the destination is a flow control peripheral or not.

21.15.3.5 Last Source Transaction Request Register (AHBDMA0_LstSrcReg)

AHDBMA0_LstSrcReg[31] is ignored when software handshaking is not enabled for the source of channel 0, or when the source of channel 0 is not a flow controller.

A channel LSTSRC bit is written only if the corresponding channel write enable bit in the LSTSRC_WE field is asserted on the same AHB write transfer.

Figure 21-73. Last Source Transaction Request Register (AHBDMA0_LstSrcReg)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	LSTSRC_WE	W	Source last transaction request write enable 0 = Write disabled 1 = Write enabled
24:30		N/A	Reserved
31	LSTSRC	R/W	Source last transaction request 0 = Not last transaction in current block 1 = Last transaction in current block

21.15.3.6 Last Destination Transaction Request Register (AHBDMA0_LstDstReg)

AHBDMA0_LstDstReg[31] is ignored when software handshaking is not enabled for the destination of channel 0 or when the destination of channel 0 is not a flow controller.

A channel LSTDST bit is written only if the corresponding channel write enable bit in the LSTDST_WE field is asserted on the same AHB write transfer.

Figure 21-74. Last Destination Transaction Request Register (AHBDMA0_LstDstReg)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	LSTDST_WE	W	Destination last transaction request write enable 0 = Write disabled 1 = Write enabled
24:30		N/A	Reserved
31	LSTDST	R/W	Destination last transaction request 0 = Not last transaction in current block 1 = Last transaction in current block

21.15.4 Miscellaneous SATA DMA Registers

This section describes miscellaneous DMA registers.

21.15.4.1 SATA DMA Configuration Register (AHBDMA0_DmacfgReg)

This register is used to enable the SATA DMA, which must be done before any channel activity can begin.

Figure 21-75. SATA DMA Configuration Register (AHBDMA0_DmacfgReg)

Bits	Name	R/W	Description
0:30		N/A	Reserved
31	DMA_EN	R/W	SATA DMA Enable bit. 0 = SATA DMA Disabled 1 = SATA DMA Enabled.

If the global channel enable bit is cleared while any channel is still active, then AHBDMA0_DmaCfgReg[DMA_EN] still returns 1 to indicate that there are channels still active until hardware has terminated all activity on all channels, at which point the AHBDMA0_DmaCfgReg[DMA_EN] bit returns 0. For more information, refer to *Abnormal Transfer Termination* on page 716.

21.15.4.2 SATA DMA Channel Enable Register (AHBDMA0_ChEnReg)

This is the SATA DMA Channel Enable Register. If software needs to set up a new channel, then it can read this register in order to find out which channels are currently inactive; it can then enable an inactive channel with the required priority.

Figure 21-76. SATA DMA Channel Enable Register (AHBDMA0_ChEnReg)

Bits	Name	R/W	Description
0:22		N/A	Reserved
23	CH_EN_WE	W	Channel enable write enable.
24:30		N/A	Reserved
31	CH_EN	R/W	Enables/Disables the channel. Setting this bit enables a channel; clearing this bit disables the channel. 0 = Disable the Channel 1 = Enable the Channel The AHBDMA0_ChEnReg.CH_EN bit is automatically cleared by hardware to disable the channel after the last AMBA transfer of the DMA transfer to the destination has completed. Software can therefore poll this bit to determine when this channel is free for a new DMA transfer.

All bits of this register are cleared to 0 when the global SATA DMA channel enable bit, AHBDMA0_DmaCfgReg[31], is 0. When the global channel enable bit is 0, then a write to the register is ignored and a read will always read back 0.

The channel enable bit, AHBDMA0_ChEnReg[CH_EN], is written only if the corresponding channel write enable bit, AHBDMA0_ChEnReg[CH_EN_WE], is asserted on the same AHB write transfer. For example, writing 0x01x1 writes a 1 into AHBDMA0_ChEnReg[31], while AHBDMA0_ChEnReg[24:30] remains unchanged. Writing 0x00*xx* leaves AHBDMA0_ChEnReg[24:31] unchanged. Note that a read-modified write is not required.

For information on software disabling a channel by writing 0 to AHBDMA0_ChEnReg.CH_EN, refer to *Disabling a Channel Prior to Transfer Completion* on page 715.

21.15.4.3 DMA ID Register (AHBDMA0_DmaIdReg)

This is the SATA DMA ID register, which is a read only register that reads back the hard coded ID number, 0x0.

Figure 21-77. DMA ID Register (AHBDMA0_DmaIdReg)

Bits	Name	R/W	Description
0:31	DMA_ID	R	Hard coded SATA DMA Peripheral ID

21.15.4.4 DMA Test Register (AHBDMA0_DmaTestReg))

This register is used to put the AHB slave interface into test mode, during which the read back value of the writable registers match the value written, assuming the SATA DMA configuration has not optimized the same registers. In normal operation, the read back value of some registers is a function of the SATA DMA state and does not match the value written.

Figure 21-78. DMA Test Register (AHBDMA0_DmaTestReg)

Bits	Name	R/W	Description
0:30		N/A	Reserved.
31	TEST_SLV_IF	R/W	Puts the AHB slave interface into test mode. In this mode, the read back value of the writable registers always matches the value written. This bit does not allow writing to read only registers. 0 = Normal mode 1 = Test mode

21.15.4.5 DMA Component Parameters Reg 2 Low (AHBDMA0_DMA_COMP_PARAMS_2_L)

This is a constant read only register that contains encoded information about the component parameter settings for Channel 0.

Figure 21-79. DMA Component Parameters Reg 2 Low (AHBDMA0_DMA_COMP_PARAMS_2_L)

Bits	Name	R/W	Description
0		N/A	Reserved
1:3	CH0_FIFO_DEPTH	R	The value of this register is derived from the DMAH_CH0_FIFO_DEPTH parameter. 0x4 = 128
4:6	CH0_SMS	R	The value of this register is derived from the DMAH_CH0_SMS parameter. 0x4 = NO_HARDCODE
7:9		N/A	Reserved (Reads back as 0x4.)
10:12	CH0_DMS	R	The value of this register is derived from the DMAH_CH0_DMS parameter. 0x4 = NO_HARDCODE
13:15	CH0_MAX_MULT_SIZE	R	The value of this register is derived from the DMAH_CH0_MULT_SIZE parameter. 0x3 = 32
16:17	CH0_FC	R	The value of this register is derived from the DMAH_CH0_FC parameter. 0x0 = DMA
18	CH0_HC_LLP	R	The value of this register is derived from the DMAH_CH0_HC_LLP parameter. 0x0 = False
19	CH0_CTL_WB_EN	R	The value of this register is derived from the DMAH_CH0_CTL_WB_EN parameter. 0x0 = False

Figure 21-79. DMA Component Parameters Reg 2 Low (AHBDMA0_DMA_COMP_PARAMS_2_L)			
Bits	**Name**	**R/W**	**Description**
20	CH0_MULTI_BLK_EN	R	The value of this register is derived from the DMAH_CH0_MULTI_BLK_EN parameter. 0x1 = True
21	CH0_LOCK_EN	R	The value of this register is derived from the DMAH_CH0_LOCK_EN parameter. 0x0 = False
22	CH0_SRC_GAT_EN	R	The value of this register is derived from the DMAH_CH0_SRC_GAT_EN parameter. 0x1 = True
23	CH0_DST_SCA_EN	R	The value of this register is derived from the DMAH_CH0_DST_SCA_EN parameter. 0x1 = True
24	CH0_STAT_SRC	R	The value of this register is derived from the DMAH_CH0_STAT_SRC parameter. 0x0 = False
25	CH0_STAT_DST	R	The value of this register is derived from the DMAH_CH0_STAT_DST parameter. 0x0 = False
26:28	CH0_STW	R	The value of this register is derived from the DMAH_CH0_STW parameter. 0x3 = 32
29:31	CH0_DTW	R	The value of this register is derived from the DMAH_CH0_DTW parameter. 0x3 = 32

21.15.4.6 DMA Component Parameters Reg 2 High (AHBDMA0_DMA_COMP_PARAMS_2_H)

This is a constant read only register that contains encoded information about the component parameter settings for Channel 0.

Figure 21-80. DMA Component Parameters Reg 2 High (AHBDMA0_DMA_COMP_PARAMS_2_H)			
Bits	**Name**	**R/W**	**Description**
0:27		NA	Reserved (Reads back as 0x3333333.)
28:31	CH0_MULTI_BLK_TYPE	R	The values of these bit fields are derived from the DMAH_CH0_MULTI_BLK_TYPE parameter. 0x0 = NO_HARDCODE

21.15.4.7 DMA Component Parameters Reg 1 Low (AHBDMA0_DMA_COMP_PARAMS_1_L)

This is a constant read only register that contains encoded information about the maximum block size parameters for Channel 0.

Figure 21-81. DMA Component Parameters Reg 1 Low (AHBDMA0_DMA_COMP_PARAMS_1_L)

Bits	Name	R/W	Description
0:27		NA	Reserved
28:31	CH0_MAX_BLK_SIZE	R	The values of these bit fields are derived from the DMAH_CH0_MAX_BLK_SIZE parameter. 0xA = 4095

21.15.4.8 DMA Component Parameters Reg 1 High (AHBDMA0_DMA_COMP_PARAMS_1_H)

This is a constant read only register that contains encoded information about the maximum block size parameters for Channel 0.

Figure 21-82. DMA Component Parameters Reg 1 High (AHBDMA0_DMA_COMP_PARAMS_1_H)

Bits	Name	R/W	Description
0:1		N/A	Reserved
2	STATIC_ENDIAN_SELECT	R	The value of this register is derived from the DMAH_STATIC_ENDIAN_SELECT parameter. 0 = False
3	ADD_ENCODED_PARAMS	R	The value of this register is derived from the DMAH_ADD_ENCODED_PARAMS parameter. 1 = False
4:8	NUM_HS_INT	R	The value of this register is derived from the DMAH_NUM_HS_INT parameter. 0x01 = 1
9:12		NA	Reserved
13:14	M2_HDATA_WIDTH	R	The value of this register is derived from the DMAH_M2_HDATA_WIDTH parameter. 0x0 = 32 bits
15:16	M1_HDATA_WIDTH	R	The value of this register is derived from the DMAH_M1_HDATA_WIDTH parameter. 0x0 = 32 bits
17:18	S_HDATA_WIDTH	R	The value of this register is derived from the DMAH_S_HDATA_WIDTH parameter. 0x0 = 32 bits
19:20	NUM_MASTER_INT	R	The value of this register is derived from the DMAH_NUM_MASTER_INT parameter. 0x1 = 2

Figure 21-82. DMA Component Parameters Reg 1 High (AHBDMA0_DMA_COMP_PARAMS_1_H)			
Bits	**Name**	**R/W**	**Description**
21:23	NUM_CHANNELS	R	The value of this register is derived from the DMAH_NUM_CHANNELS parameter. 0x0 = 1
24:27		N/A	Reserved
28	MABRST	R	The value of this register is derived from the DMAH_MABRST parameter. 0 = False
29:30	INTR_IO	R	The value of this register is derived from the DMAH_INTR_IO parameter. 0x2 = COMBINED
31		N/A	Reserved

21.15.4.9 DMA Component Type Register (AHBDMA0_DMA_COMP_TYPE)

This is the SATA DMA Component Type register, which is a read only register that specifies the component type.

Figure 21-83. DMA Component Type Register (AHBDMA0_DMA_COMP_TYPE)			
Bits	**Name**	**R/W**	**Description**
0:31	DMA_COMP_TYPE	R	Component type number: 0x4457_1110.

21.15.4.10 DMA Component Version Register (AHBDMA0_DMA_COMP_VER)

This is the SATA DMA Component Version register, which is a read only register that specifies the version of the packaged component.

Figure 21-84. DMA Component Version Register (AHBDMA0_DMA_COMP_VER)			
Bits	**Name**	**R/W**	**Description**
0:31	DMA_COMP_VERSION	R	Version of the component: 0x3231_302A (ASCII for 210*, Version 2.10*.

22. Double Data Rate (DDR) SDRAM Controller

The PPC460EX/EXr/GT memory subsystem consists of the following primary components:

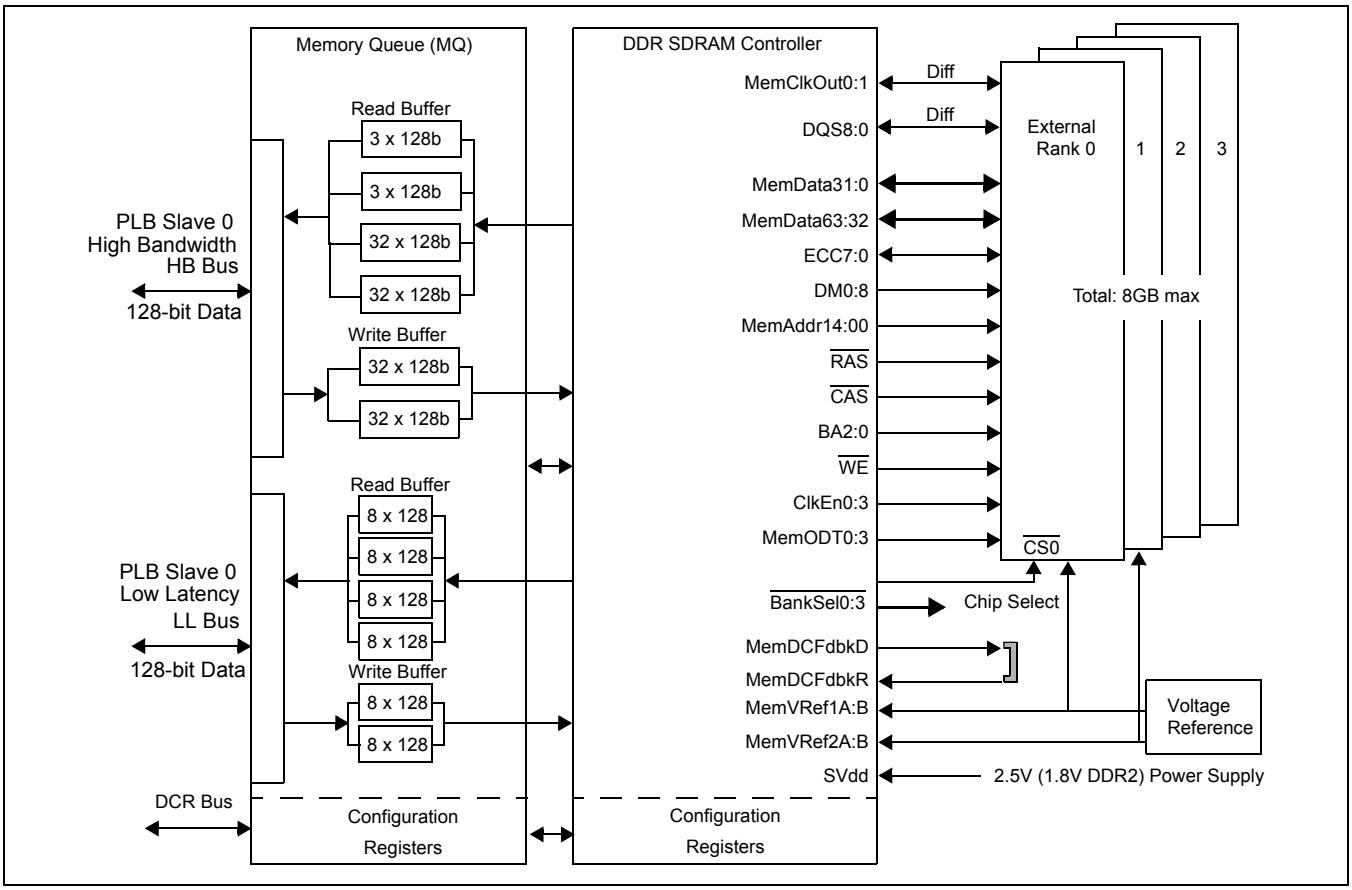
- Memory Queue Module (MQ)
- Double Data Rate (DDR) SDRAM Controller

The Memory Queue is the interface to the internal local buses: PLB Slave Segment 0 and PLB Slave Segment 1. It contains buffers, status registers and configuration registers for the PLB interfaces. Registers associated with the Memory Queue have the prefix MQ0.

The DDR SDRAM Controller is the interface to the DDR memory. It contains status and configuration registers specific to the memory interface. Registers associated with the DDR SDRAM Controller or memory controller interface have the MCIF0 prefix.

Figure 22.1 shows a block diagram of these components.

Figure 22-1. Memory Subsystem



22.1 Memory Queue Module (MQ)

The memory queue is a high-performance transaction queue with large buffers that connects the DDR1 and 2 memory controller to two PLB slave ports of the 128-bit wide Processor Local Bus.

The memory queue module provides two parallel paths from PLB to memory. Each path has a two-deep posted write-submit queue, a four-deep read-submit queue and a four-deep read-completion queue. One path is designated as High Bandwidth (HB) and the other as Low Latency (LL). The HB path stores data in two 512B write FIFOs and four 512B read FIFOs. The LL path stores data in two 128B write FIFOs and four 128B read FIFOs.

The memory queue's PLB slave interface exploits the PLB's four-deep address-pipelining capability. It is expected that the HB Bus (the one on which the DMA controller moves data) uses "Rearbitration on First Read Request" mode, and the LL Bus (the one to which the 440 CPU is dedicated) uses "Address Acknowledgement on First Read Request" mode. If a slot is available, write requests are loaded immediately into the write submit queue. Read and write transfers may complete simultaneously over the separate read and write data buses.

The memory queue supports read passing. The read passing feature (RPEN bit in Configuration Register 1) (MQ0_CF1H), enables the processor to access memory more quickly by allowing the processor read request to pass DMA data requests. See *Figure 22-4* on page 746 for details about the RPEN bit.

A DCR interface provides access to configuration and error reporting registers.

The PLB slave connects the PLB to the queuing structures. It is designed to *PLB Specification* Version 4.4.

There are two instantiations of the PLB slave: one for the high-bandwidth PLB, the other for the low-latency PLB. The slaves are identical, with the following exceptions:

The high-bandwidth slave supports maximum burst transfers of 512B.

The low-latency slave supports maximum burst transfers of 128B.

There is no physical difference between the slaves; they have the same data width and operate at the same frequency. The type of PLB traffic that is directed to each slave, as configured in the PLB crossbar, defines the "low-latency" and "high-bandwidth" terms. The low-latency bus is expected to handle the majority of traffic from the processor because this bus has higher priority access to the memory controller than the high-bandwidth bus. Additionally, the low-latency bus should primarily receive single-beat and line transfers, while the high-bandwidth bus should process large data-mover requests from the DMA controller.

Descriptions of the "slave" for the rest of this document pertain to both slaves unless otherwise indicated.

The PLB slave interfaces to the memory queue's submit queue, completion queue, and data FIFOs. It is responsible for pushing new requests onto the submit queue and popping completed reads off of the completion queue. The queuing structures handle the completion of posted writes and the transfer of reads from the submit queue to the completion queue.

The PLB slave supports the following:

- 128-bit masters only
- MEMORY type only
- Single-beat, 4-word and 8-word line, and quad-word, fixed-length burst transfers
- Read line target word first
- Read underflow
- Read and write aborts
- Read wait states only when Read Flow Through is enabled

- 2-deep, 3-deep, and 4-deep read pipelining
- 2-deep write pipelining
- Read and write data flowing simultaneously over the separate PLB read and write data buses
- 3-cycle PLB acknowledge arbitration

The PLB slave does *not* support the following:

- Variable-length burst transfers
- Locked transactions
- 16-word line transfers
- PLB attributes
- Read and write overflow. The slave disconnects fixed-length burst transfers at 512B (high-bandwidth) or 128B (low-latency) or at the end of a physical bank.
- Read wait states when Read Flow Through is disabled
- Write wait states

22.1.1 Memory Queue Module Registers

The memory queue's registers are accessed by means of the DCR bus with address range 0x0040 through 0x004F in the PPC460EX/EXr/GT.

The configuration values must be set immediately following reset and remain unchanged. Modifying the configuration registers after normal operation has begun is not supported and could have unpredictable results.

Write cycles to reserved addresses have no effect on the system. Writes complete normally on the DCR bus, however no register values are modified as a result. Read cycles to reserved addresses return a value of 0.

22.1.1.1 Memory Queue Register Map

The following table lists the memory queue registers.

Table 22-1. Memory Queue Register Map

Mnemonic	Register	DCR Address	Access	Page
MQ0_B0BAS	Rank 0 Base Address and Size	0x0040	R/W	744
MQ0_B1BAS	Rank 1 Base Address and Size	0x0041	R/W	744
MQ0_B2BAS	Rank 2 Base Address and Size	0x0042	R/W	744
MQ0_B3BAS	Rank 3 Base Address and Size	0x0043	R/W	744
MQ0_CF1H	Configuration 1 (HB)	0x0045	R/W	746
MQ0_ESH	Error Status (HB)	0x0047	R/W	747
MQ0_EAUH	Error Address, Upper 32 Bits (HB)	0x0048	R	747
MQ0_EALH	Error Address, Lower 32 Bits (HB)	0x0049	R	748
MQ0_BAUL	PLB Base Address, Upper 32 Bits (LL)	0x004A	R/W	748
MQ0_CF1L	Configuration 1 (LL)	0x004B	R/W	749
MQ0_ESL	Error Status (LL)	0x004C	R/W	749
MQ0_EAUL	Error Address, Upper 32 Bits (LL)	0x004D	R	750
MQ0_EALL	Error Address, Lower 32 Bits (LL)	0x004E	R	750
MQ0_CFBHL	Configuration Between HB and LL Paths	0x004F	R/W	751
MQ0_BAUH	PLB Base Address, Upper 32 Bits (HB)	0x0050	R/W	745

22.1.1.2 Rank n Base Address and Size Register (MQ0_BnBAS)

MQ0_BnBAS contains the base address and size of physical memory rank *n*.

Figure 22-2. Rank n Base Address and Size Register (MQ0_BnBAS)

0:10	BA	Rank n Base Address	Base address of physical memory rank *n*. Valid base address if there is a 1 anywhere in the size field [16:27]. These 11 bits of the base address [0:10] are the upper bits of the 34 bits physical address. If the SIZE field [16:27] is zero, access to rank *n* is disabled. (See table notes.) The minimum Bank size is 8MB.
11:15		Reserved. Must be left at 00000.	

16:27	SIZE	Rank n ($\overline{\text{BankSel}}$) Size 0x000 0 MB 0xFFC 8 MB 0xFF8 16 MB 0xFF0 32 MB 0xFE0 64 MB 0xFC0 128 MB 0xF80 256 MB 0xF00 512 MB 0xE00 1024 MB 0xC00 2048 MB 0x800 4096 MB	Valid size of physical memory rank *n*, from 8 MB to 4 GB. A size of zero disables access to rank *n*. Bits [17:25] correspond to base address bits [0:8]. Note: The maximum total memory size of all ranks must not exceed 8 GB. SIZE = [2^(#row bits + #column bits)]*(#internal banks)*(bus width of memory chip in bytes)*(#memory chips per rank)
28:31		Reserved	
Notes: 1. The base address must be a multiple of the size. 2. If different sized ranks are installed, they must be installed largest to smallest, beginning with rank 0.			

The DDR controller utilizes 34-bits of the 36-bit PLB address. The most significant 2 bits of the 36-bit PLB address (bits 28 and 29 of the upper 32 bit address) allow the 34-bit address to have an alias on the low latency and high bandwidth PLB slave segments. Accesses with the upper two bits set to 0b00 are made over the low latency (LL) slave interface and accesses with the upper two bits set to 0b10 are made over the high bandwidth (HB) slave interface.

The 11-bit base address bit field, MQ0_BnBAS[BA], includes bits 30:31 of the upper 32-bit address and bits 0:9 of the lower 32-bit address. The following table identifies the address bits defined by the 11-bit base address.

Table 22-2. PLB Address Format

	Upper 32-bit PLB Address			**Lower 32-bit PLB Address**
	0 27	28 31	0	31
		36-bit PLB Address		
LL		00	BA (11-bits)	00000000000000000000
HB		10	BA (11-bits)	00000000000000000000

22.1.1.3 PLB Base Address Register Upper 32 Bits (HB) (MQ0_BAUH)

MQ0_BAUH contains the 32 most significant bits of the PLB base address for the high bandwidth path between the PLB and memory.

Figure 22-3. PLB Base Address Register Upper 32 Bits (HB) (MQ0_BAUH)			
0:31	BA	Upper 32 bits of PLB Base Address	Each SDRAM memory byte is accessible in two 4-GB spaces; one Low Latency (LL) and one High Bandwidth (HB) PLB slave. 0x 0000 0008 must be set if High Bandwidth (HB) PLB access is required.

22.1.1.4 Configuration Register 1 (HB) (MQ0_CF1H)

MQ0_CF1H contains partial configuration information for the high bandwidth path between the PLB and memory.

Reset value = 0x0000_1080.

Figure 22-4. Configuration Register 1 (HB) (MQ0_CF1H)

0	AAFR*	Address Acknowledgement on First Request	Causes the slave to assert PLB slave address acknowledge signal the first time a read request is presented on PLB, instead of asserting a PLB slave rearbitrate signal.
1:11		Reserved	
12	PRPD	PLB Read Pipelining Disable	Disables PLB pipelining for read cycles. *Intended for debug purposes only; setting this bit degrades performance.*
13	PWPD	PLB Write Pipelining Disable	Disables PLB pipelining for write cycles. *Intended for debug purposes only; setting this bit degrades performance.*
14	PRW	PLB Read Wait (do not rearbitrate)	Causes the PLB slave to assert a PLB slave wait signal, rather than PLB slave rearbitrate signal, when it cannot immediately service a read request. Read pipelining is disabled, regardless of the value of bit 12. PLB will be stalled until the read completes on the memory bus. *Intended for debug purposes only, as setting this bit degrades performance.*
15		Reserved	
16:19	RPLM	Read Passing Limit	Indicates the maximum number of consecutive times that a read is allowed to pass eligible transactions within the same data path (low latency or high bandwidth). Default is one time. A value of zero indicates that there is no limit on the number of times that a read can pass other transactions. Bit 20 must be enabled in order for this value to be significant.
20	RPEN	Read Passing Enable	Enables the read passing feature.
21	RFTE	Read Flow Through Enable	Allows the MQ read FIFOs to flow data through from MCIF to PLB. Default operation is bucket mode, for which all read data has completed on MCIF before the read transaction starts on PLB.
22:24	WRCL	MCIF Cycle Limit	When a write has been passed, indicates the maximum number of consecutive MCIF (Memory Controller) cycles for the write request that will be run before allowing another read to pass.
25:31		Reserved	

Note: If PLB Read Wait, the Do Not Rearbitrate bit (bit 14) is enabled; it overrides Address Acknowledge on first Request. On the first read request, the PLB slave wait signal is asserted.

22.1.1.5 Error Status Register (HB) (MQ0_ESH)

MQ0_ESH contains error status information for the high bandwidth path between the PLB and memory.

Figure 22-5. Error Status Register (HB) (MQ0_ESH)

0:11	EMID	Error PLB Master ID bit 0 Master #0 ICU Read 1 Master #1 DCU Read 2 Master #2 PCI0 3 Master #3 PCIE1 or SRIO 4 Master #4 DCU Write 5 Master #5 PCIE0 6 Master #6 DMA (DMA2P40) 7 Master #7 I2O/DMA (HSDMA) 8 Master #8 I2O/DMA (I2O/FIFO) 9 Master #9 Security Engine (EIP94) 10 Master #10 MAL 11 Master #11 PLB4-to-AHB bridge (PLB4AHB)	One bit per master. A value of 1 in any of these bits indicates that the corresponding PLB master experienced an error during a read or write from the Memory Queue. Bits 12:14 indicate the type of error that occurred. The set bit is cleared by writing a 1 to the respective bit position.
12	UEER	Uncorrectable ECC Error*	Indicates that an uncorrectable ECC error was reported during an MCIF Bus operation. This bit is set coincident with the setting of one of the master bits 0:11 of this register. Clearing the Error Master ID (bits 0:11 of this register) clears this bit.
13	WTER	Write Transaction Error	Indicates that the error latched occurred during a write transaction. Clearing the Error Master ID (bits 0:11 of this register) clears this bit.
14	RTER	Read Transaction Error	Indicates that the error latched occurred during a read transaction. Clearing the Error Master ID (bits 0:11 of this register) clears this bit.
15:31		Reserved	

Note: Correctable ECC errors are handled by the AppliedMicro memory controller.

Note: Only the *first* of multiple errors is latched and held until the Error Master ID bit corresponding to the PLB master of the erroneous transaction is cleared. Only one bit of the EMID field can be set at any one time.

The PLB address corresponding to the latched error is recorded in the Error Address registers. See *Error Detection and Error Handling* on page 809 for details.

22.1.1.6 Error Address Register Upper 32 Bits (HB) (MQ0_EAUH)

MQ0_EAUH contains the 32 most significant bits of the PLB address for a transaction failure during an MCIF bus operation on the high bandwidth path between the PLB and memory.

Figure 22-6. Error Address Register Upper 32 Bits (HB) (MQ0_EAUH)

0:31	EA	Error Address	Contains the most significant 32 bits of the PLB address of an erroneous transaction that occurred during an MCIF Bus operation. This register is set coincident with the setting of one of the Error Master ID bits (0:11 of the Error Status register Clearing bits 0:11 of the Error Status register clears this register.

22.1.1.7 Error Address Register Lower 32 Bits (HB) (MQ0_EALH)

MQ0_EALH contains the 32 least significant bits of the PLB address for a transaction failure during an MCIF bus operation on the high bandwidth path between the PLB and memory.

Figure 22-7. Error Address Register Lower 32 Bits (HB) (MQ0_EALH)

0:31	EA	Error Address	Contains the least significant 32 bits of the PLB address of an erroneous transaction that occurred during an MCIF Bus operation. This register is set coincident with the setting of one of the Error Master ID bits (0:11 of the Error Status register Clearing bits 0:11 of the Error Status register clears this register.

22.1.1.8 PLB Base Address Register Upper 32 Bits (LL) (MQ0_BAUL)

MQ0_BAUL contains the 32 most significant bits of the PLB base address for the low latency path between the PLB and memory.

Figure 22-8. PLB Base Address Register Upper 32 Bits (LL) (MQ0_BAUL)

0:31	BA	Upper 32-bits of PLB Base Address	Corresponds to the 32 most significant bits of the PLB base address. This should be set to 0x 0000 0000.

22.1.1.9 Configuration Register 1 (LL) (MQ0_CF1L)

MQ0_CF1L contains partial configuration information about the low latency path between the PLB and memory.

Reset value = 0x0000_1080.

Figure 22-9. Configuration Register 1 (LL) (MQ0_CF1L)

0	AAFR	Address Acknowledgement on First Request	Causes the slave to assert PLB slave address acknowledge signal the first time a read request is presented on PLB, instead of asserting a PLB slave rearbitrate signal.
1:11		Reserved	
12	PRPD	PLB Read Pipelining Disable	Disables PLB pipelining for read cycles. *For debug purposes only; setting this bit degrades performance*
13	PWPD	PLB Write Pipelining Disable	Disables PLB pipelining for write cycles. *For debug purposes only; setting this bit degrades performance.*
14	PRW	PLB Read Wait (do not rearbitrate)	Causes the PLB slave to assert a PLB slave wait signal, rather than PLB slave rearbitrate signal, when it cannot immediately service a read request. Read pipelining is disabled, regardless of the value of bit 12. PLB will be stalled until the read completes on the memory bus. *Intended for debug purposes only, as setting this bit degrades performance.*
15		Reserved	
16:19	RPLM	Read Passing Limit	Indicates the maximum number of consecutive times that a read is allowed to pass eligible transactions within the same data path (low-latency or high-bandwidth). Default is 1 time. A value of zero indicates that there is no limit on the number of times a read can pass other transactions. Bit 20 must be enabled for this value to be significant.
20	RPEN	Read Passing Enable	Enables the read passing feature.
21	RFTE	Read Flow Through Enable	Allows the MQ read FIFOs to flow data through from MCIF to PLB. Default operation is bucket mode, in which all read data has completed on the DDR memory controller interface before the read transaction starts on PLB.
22:31		Reserved	

Note: If PLB Read Wait, the Do Not Rearbitrate bit (14) is enabled; it overrides Address Acknowledge on first Request. On the first read request, the PLB slave wait signal is asserted.

22.1.1.10 Error Status Register (LL) (MQ0_ESL)

MQ0_ESL contains error status information for the low latency path between the PLB and memory. The following figure describes the MQ0_ESL register bits.

Figure 22-10. Error Status Register (LL) (MQ0_ESL)

0:11	EMID	Error PLB Master ID	One bit per master; a value of 1 in any of these bits indicates that the corresponding PLB master experienced an error during a read or write from the Memory Queue. Bits 12:14 indicate the type of error that occurred. The set bit is cleared by writing a 1 to the respective bit position.

12	UEER	Uncorrectable ECC Error	Indicates that an uncorrectable ECC error was reported during an MCIF Bus operation. This bit is set coincident with the setting of one of the master bits 0:11 of this register. Clearing the Error Master ID (bits 0:11 of this register) clears this bit.
13	WTER	Write Transaction Error	Indicates that the error latched was caused during a write transaction. Clearing the Error Master ID (bits 0:11 of this register) clears this bit.
14	RTER	Read Transaction Error	Indicates that the error latched was caused during a read transaction. Clearing the Error Master ID (bits 0:11 of this register) clears this bit.
15:31		Reserved	
Correctable ECC errors are handled by the AppliedMicro memory controller.			

Note: Only the *first* of multiple errors is latched and held until the Error Master ID bit corresponding to the PLB master of the erroneous transaction is cleared. Only one bit of the EMID field can be set at any one time.

The PLB address corresponding to the latched error is recorded in the Error Address Registers. See *Section 22.1.1.11 Error Address Register Upper 32 Bits (LL) (MQ0_EAUL)* and *Section 22.1.1.12 Error Address Register Lower 32 Bits (LL) (MQ0_EALL)* for details.

22.1.1.11 Error Address Register Upper 32 Bits (LL) (MQ0_EAUL)

MQ0_EAUL contains the most significant 32 bits of the PLB address of an erroneous transaction that occurred during an MCIF Bus operation on the low latency path between the PLB and memory. This register is set coincident with the setting of one of the Error Status register Error Master ID bits (MQ0_ESL[EMID]).

Figure 22-11. Error Address Register Upper 32 Bits (HB) (MQ0_EAUL)

0:31	EA	Error Address bits [0:31]	Contains the most significant 32 bits of the PLB address of an erroneous transaction that occurred during an MCIF Bus operation. This register is set coincident with the setting of one of the Error Master ID bits 0:11 of the Error Status register. Clearing bits 0:11 of the Error Status register clears this register.

22.1.1.12 Error Address Register Lower 32 Bits (LL) (MQ0_EALL)

MQ0_EALL contains the least significant 32 bits of the PLB address of an erroneous transaction that occurred during an MCIF Bus operation on the low latency path between the PLB and memory. This register is set coincident with the setting of one of the Error Status register Error Master ID bits (MQ0_ESL[EMID]).

Figure 22-12. Error Address Register Lower 32 Bits (LL) (MQ0_EALL)

0:31	EA	Error Address bits [32:63]	Contains the least significant 32 bits of the PLB address of an erroneous transaction that occurred during an MCIF Bus operation. This register is set coincident with the setting of one of the Error Master ID bits 0:11 of the Error Status register. Clearing bits 0:11 of the Error Status register clears this register.

22.1.1.13 Configuration Between HB and LL Paths Register (MQ0_CFBHL)

MQ0_CFBHL contains a variety of information about transactions that involve both the high bandwidth path and the low latency path between the PLB and memory.

Reset value = 0x10A6_8000.

Figure 22-13. Configuration Between HB and LL Paths Register (MQ0_CFBHL)

0:3	TPLM	Transaction Passing Limit	Indicates the maximum number of consecutive times a transaction (read or write) in the low-latency path is allowed to pass eligible transactions in the high-bandwidth path. Default is 1 time. A value of zero indicates there is no limit on the number of times a read can pass other transactions. Bit 4 must be enabled for this value to be significant.
4	TPEN	Transaction Passing Enable	Enables transaction passing feature.
5	MQST	MQ Stop	Allows the MQ to be HALTED from issuing commands on MCIF. Queuing is still allowed on the PLB. *For Test and Lab use only.* Allows testing of specific configurations of queued requests.
6:8	HBCL	MCIF Cycle Limit	When an HB request has been passed, indicates the maximum number of consecutive memory controller cycles for the HB request that runs before allowing another LL request.
9:30		Reserved	
31	H2OMW	H2O MQ Watermark	Used for revision control in releases of netlists to AppliedMicro. The value of this bit is irrelevant. This bit's name is updated in the netlist to denote the current netlist revision level.

22.2 Double Data Rate (DDR) SDRAM Controller

The DDR memory controller supports DDR 1 (double data rate) and DDR 2 SDRAMs. It consists of an MCIF2 interface and a DCR interface, and provides an external DDR1/DDR 2 SDRAM interface with optional ECC capability.

Industry standard DIMMS are supported, allowing for a variety of system memory configurations. Up to four logical ranks are supported in limited configurations, supporting up to 8GB. Global memory timings, address and rank sizes, and memory addressing modes are programmable. Configuration of all DDR SDRAM registers is performed through the DCR slave port.

The DDR SDRAM memory controller provides the following support.

22.2.1 MCIF Interface

MCIF support includes:

- Four-deep request queue
- 128-bit read and write data paths
- Programmable address compare for each memory rank
- 64 GB-aligned address space within 36-bit address space

22.2.2 DDR 1 and DDR 2 SDRAM

DDR 1 and DDR 2 SDRAM support include:

- Registered and nonregistered industry standard DIMMs
- 64/72-bit and 32/40-bit memory interface with optional ECC
- 1-4 chip selects
- Burst length of 4.
- DDR1 CAS latencies of 2, 2.5, and 3 supported
- DDR2 CAS latencies of 2, 3, 4, 5, 6, and 7 supported
- Up to 200MHz/400Mbps double data rate memory (DDR2 400).
- Page mode accesses (Up to 32 open pages) with configurable paging policy
- Look-ahead request queue with programmable depth of 4 commands.
- Optimized command scheduling (activate/precharge non-conflicting ranks while accessing the current rank)
- 8 MB to 2 GB; 4- or 8-bank devices
- Up to 2 GB per rank
- Support for controller and memory on-die termination (ODT)
- Programmable address mapping and timing
- Auto refresh (CBR)
- Synchronize SDRAM configuration by means of Mode Register Set and Extended Mode Register Set commands

22.2.3 ECC

ECC support includes:

- Standard SEC/DED coverage
- Aligned nibble error detect
- Bypass mode (disable)

22.2.4 Power Management

Power management support includes software- and hardware-initiated self-refresh.

22.2.5 Interface Signals

The PPC460EX/EXr/GT uses PowerPC bit ordering (bit 0 is the most significant bit). However, the DDR SDRAM address signals (MemAddr14:0 and BA2:0), and external data bus (Memdata[63:0]) use industry-standard bit ordering (bit 0 is the least significant bit). When a data word is written to the DDR memory interface, the most significant internal bit is 0 and appears as the least significant bit on the DDR interface. The remaining bits of the word follow in order. The main effect of this is that when data internal to the PPC460EX/EXr/GT is written to the DDR interface, the bits appear reversed. For example, an internal register value of 0xF753 corresponds to a value of 0xCAEF on the external memory data. *Figure 22-1* on page 741 shows the DDR SDRAM signal I/Os.

Table 22-3 shows the usage and signal state during and after a chip reset for each of the DDR SDRAM controller signals.

Table 22-3. DDR SDRAM Signal Usage and State During/Following Reset

Signal	During Reset State	Following Reset State	Usage
ClkEn3:0	0s	0s	Clock Enable.
BA2:0	0s	0s	Bank Address. Used to select an internal SDRAM bank in 2-4-8 bank SDRAM devices.
MemAddr14:0	0s	0s	Memory Address. See *SDRAM Commands and Operations* on page 799 for details.
$\overline{RAS}$	1	1	Row Address Strobe.
$\overline{CAS}$	1	1	Column Address Strobe.
$\overline{WE}$	1	1	Write Enable.
MemData31:0	Unknown	High-Z	Data input/output. MemData0 is the least significant bit.
MemData63:32	Unknown	High-Z	Data input/output. MemData63 is the most significant bit.
DQS8:0	Unknown	High-Z	Byte Lane Strobe.
ECC7:0	Unknown	High-Z	ECC Check Bits.
DM8:0	1s	1s	Byte Lane Mask.
$\overline{BankSel0:3}$	1s	1s	Select up to four external DDR SDRAM ranks ($\overline{CS}$ Chip Select on SDRAM)
MemODT0:3			Memory On-Die Terminator

22.2.5.1 On-Die Termination (ODT)

ODT enables performance improvements by reducing reflections on the conductors between the memory controller and SDRAM memory.

Two ODT registers control this function. One register (MCIF0_CODT) is integrated in the SDRAM. The second register is part of the memory controller. To reduce power dissipation and achieve symmetric waveforms, the terminal resistors are connected to 1/2 Vdd = Vtt.

It is possible to program the ODT differently for each type of transaction.

22.2.6 Supported Memory

The DDR SDRAM Controller supports x8 and x16 DDR1 and DDR2 memories addressable with 15 address bits; x4 memories are not supported. *Table 22-4* and *Table 22-5* list standard memory configures with the address mode, memory size per rank, and memory chips needed per rank.

The memory used for ECC check bits must support the same addressing mode as the memory for data bits. It is best to use the same memory for data and ECC to avoid potential timing differences and to guarantee the memory is large enough to contain the ECC check bits for the entire data memory.

Note:

1. ECC0:7 signals are required when operating with either a 32-bit or 64-bit DDR controller interface.
2. Unused signals included unused ECC bits do not require termination. Unused receivers are gated by the DDR SDRAM Controller.

Table 22-4. DDR1 Memory

DDR1 Memory[1]	Row x Column x Banks[2]	Addressing Mode[3]	32-bit Interface[6] (MemData0:31)		64-bit Interface[6] (MemData0:63)	
			Size/Rank[4] (MB)	Memory Chips/Rank[5]	Size/Rank[4] (MB)	Memory Chips/Rank[5]
8Mb X 16 (2Mb X 16-bit X 4 banks)	12x9x4	1	32	2	64	4
16Mb X 8 (4Mb X 8-bit X 4 banks)	12x10x4	2	64	4	128	8
16Mb X 16 (4Mb X 16-bit X 4 banks)	13x9x4	1	64	2	128	4
32Mb X 8 (8Mb X 8-bit X 4 banks)	13x10x4	2	128	4	256	8
32Mb X 16 (8Mb X 16-bit X 4 banks)	13x10x4	2	128	2	256	4
64Mb X 8 (16Mb X 8-bit X 4 banks)	13x11x4	3	256	4	512	8
64Mb X 16 (16Mb X 16-bit X 4 banks)	14x10x4	2	256	2	512	4
128Mb X 8 (32Mb X 8-bit X 4 banks)	14x11x4	3	512	4	1024	8

Note:

1. Set MCIF0_MCOPT1[DDR2]=0 for DDR1.
2. MCIF0_MCOPT1[BCNT] selects number of banks.
3. MCIF0_MBnCF[M_AM] selects the addressing mode.
4. MQ0_BnBAS[SZ] selects rank size.
5. Four ranks are supported using chip select signals, $\overline{\text{BankSel0:3}}$.
6. MCIF0_MCOPT1[WDTH] selects interface width.

Table 22-5. DDR2 Memory

DDR2 Memory[1]	Row x Column x Banks[2]	Addressing Mode[3]	32-bit Interface[6] (MemData0:31)		64-bit Interface[6] (MemData0:63)	
			Size/Rank[4] (MB)	Memory Chips/Rank[5]	Size/Rank[4] (MB)	Memory Chips/Rank[5]
16Mb X 16 (4Mb X 16-bit X 4 banks)	13x9x4	1	64	2	128	4
32Mb X 8 (8Mb X 8-bit X 4 banks)	13x10x4	2	128	4	256	8
32Mb X 16 (8Mb X 16-bit X 4 banks)	13x10x4	2	128	2	256	4
64Mb X 8 (16Mb X 8-bit X 4 banks)	14x10x4	2	256	4	512	8
64Mb X 16 (8Mb X 16-bit X 8 banks)	13x10x8	7	256	2	512	4
128Mb X 8 (16Mb X 8-bit X 8 banks)	14x10x8	7	512	4	1024	8
128Mb X 16 (16Mb X 16-bit X 8 banks)	14x10x8	7	512	2	1024	4
256Mb X 8 (32Mb X 8-bit X 8 banks)	15x10x8	7	1024	4	2048	8

Table 22-5. DDR2 Memory (Continued)

DDR2 Memory[1]	Row x Column x Banks[2]	Addressing Mode[3]	32-bit Interface[6] (MemData0:31)		64-bit Interface[6] (MemData0:63)	
			Size/Rank[4] (MB)	Memory Chips/Rank[5]	Size/Rank[4] (MB)	Memory Chips/Rank[5]
256Mb X 16 (32Mb X 16-bit X 8 banks)	15x10x8	7	1024	2	2048	4

Note:
1. Set MCIF0_MCOPT1[DDR2]=1 for DDR2.
2. MCIF0_MCOPT1[BCNT] selects number of banks.
3. MCIF0_MBnCF[M_AM] selects the addressing mode.
4. MQ0_BnBAS[SZ] selects rank size.
5. Four ranks are supported using chip select signals, $\overline{\text{BankSel0:3}}$.
6. MCIF0_MCOPT1[WDTH] selects interface width.

22.2.7 DDR SDRAM Fine Tuning

The memory clock, write signals, and read signals can all be advanced or delayed, providing many tuning options.

22.2.7.1 Delay Line (DLL) Calibration

The DDR SDRAM controller has a master DLL and several slave DLLs. The master DLL automatically adjust for variations in temperature and voltage. The slave DLLs track the master DLL in order to accurately provide the programmable fine delays set by the following bit fields: MCIF0_WRDTR[WDFD], MCIF0_RQDC[RQFD] and MCIF0_RFDC[RFOS, RFFD].

The MCIF0_DLCR register is used to calibrate the master DLL. The value in MCIF0_DLCR[DLCV] identifies the number of delay steps in a single DDR 1x half clock cycle (half one MemClkOut cycle). The DDR SDRAM controller initiates the calibration process automatically after power-on reset. See *Delay Line Calibration Register (MCIF0_DLCR)* on page 785 for details about this register.

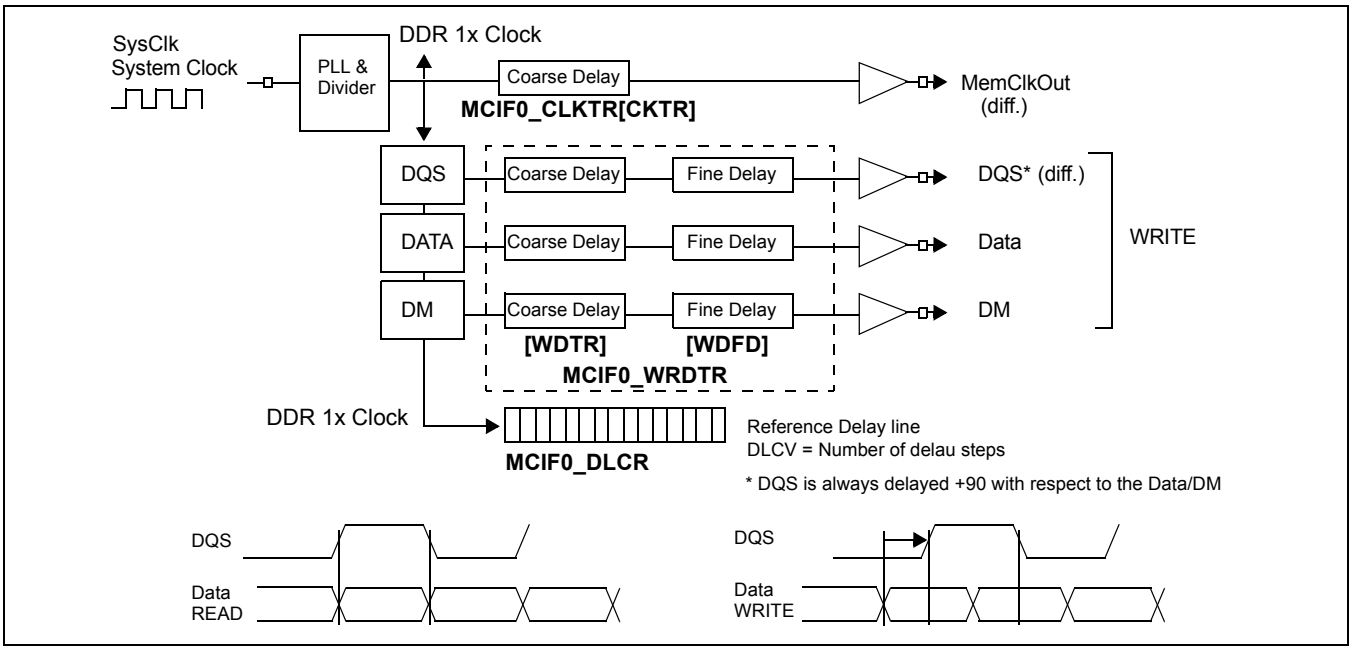
22.2.7.2 Memory Clock Timing

The MCIF0_CLKTR register can be used to advance the differential memory clock in large increments (90, 180, or 270 degrees with respect to the DDR 1x clock). See *Memory Clock Timing Register (MCIF0_CLKTR)* on page 786.

22.2.7.3 Write Data/DM/DQS Timing

The MCIF0_WRDTR register can be used to advance or delay the Data/Strobe/Mask signals (with respect to the DDR 1x clock). This makes it possible to compensate for driver and external circuit delays, and to meet various memory device requirements. See *Write Data/DM/DQS Timing Register (MCIF0_WRDTR)* on page 786 for details.

Figure 22-14. DDR SDRAM Write Circuitry



22.2.7.4 Read Data Timing

Data on a read is edge aligned with the DQS. To capture the incoming data on the rising and falling edges, DQS is delayed by the DDR controller in order to center a DQS edge on valid data. The MCIF0_RFDC, MCIF0_RDCC and MCIF0_RQDC registers are used to delay DQS and calibrate read operations.

Figure 22-15 shows the data read path of a single data bit. Data entering on the left is captured in the Stage 1 Flip Flops. The four Flip Flops in Stage 1 capture a 4-beat burst on the DDR interface. Data captured on the rising edge of DQS is stored in the even numbered Flip Flops. Likewise, data captured on the falling edge of DQS is stored in the odd numbered Flips Flops. To latch the data in Stage 1, a delayed version of DQS is used. Initialization software is responsible for calibrating the DQS delay timing (MCIF0_RQDC[RQFD]) so that DQS is centered on valid data. Since there is process variation between parts and possible voltage variations on boards, read tuning is required. Fixed DQS delay values should not be used on production systems.

The Feedback Data Capture Window selects which Flip Flop is used to store the data sampled by DQS. Each output of this block generates a pulse to an input multiplexer. The series of four pulses selecting the input multiplexer is initiated by a feedback signal pulse on the input of the Feedback Data Capture Window. The DDR controller calculates when to assert the feedback signal based on when the data should be present after a read command.

The width of the feedback pulse is one DDR 1x clock period. The internal DDR 1x clock is the same frequency as MemClkOut0:1.

The feedback pulse is generated with the rising edge of MemClkOut0 when the CAS latency expires. The pulse is driven external to the DDR controller on the feedback driver pin (MemDCFbdkD) and received on the feedback receiver pin (MemDCFbdkR). Driving the feedback pulse externally enables the DDR controller to compensate for driver/receiver timing variations due to temperature and supply voltage.

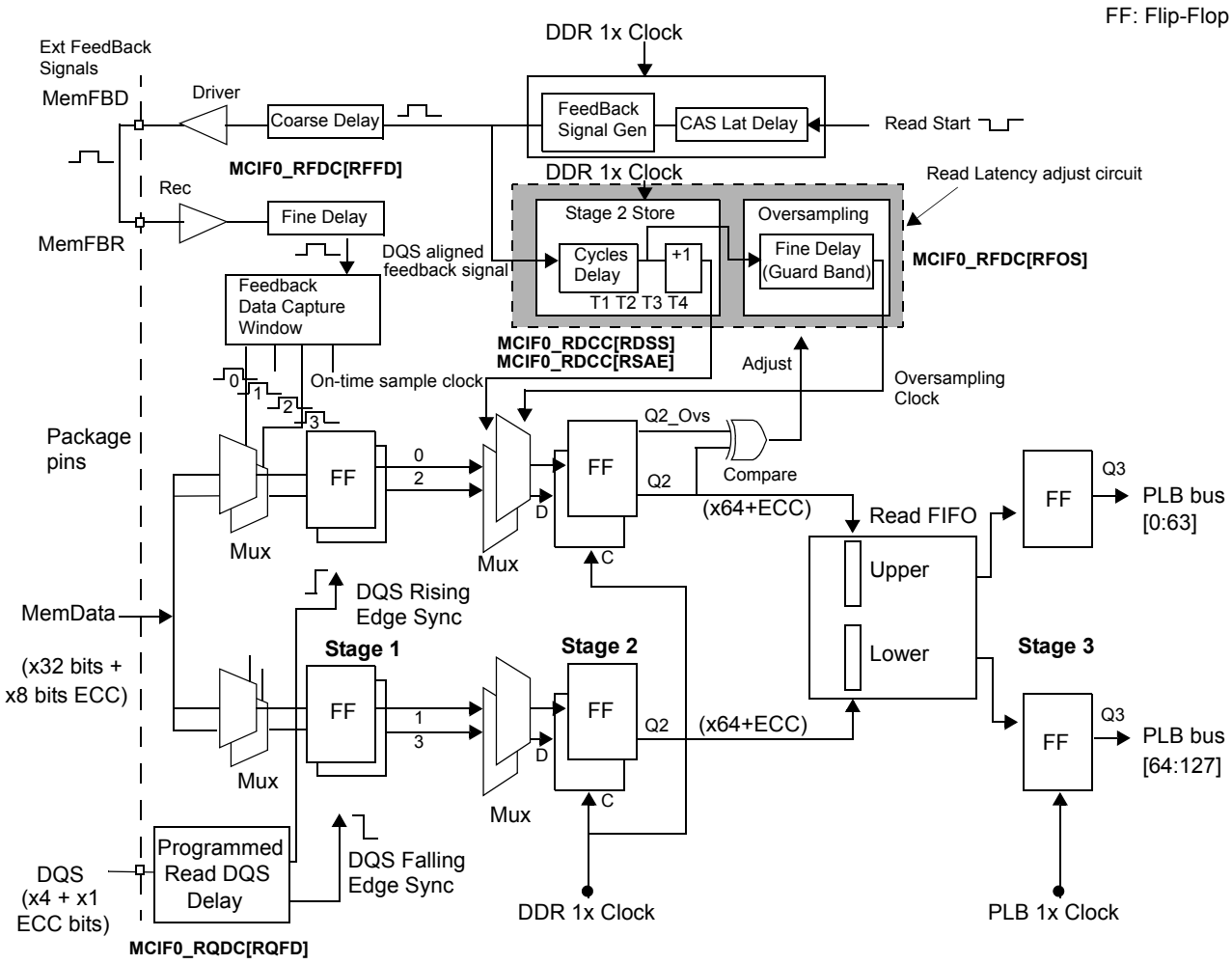
The trace connecting MemDCFbdkD to MemDCFbdkR could be length matched to the round-trip delay measured from the rising edge of MemClkOut0 to DQS on a read operation. However, it is better to connect MemDCFbdkD directly to the MemDCFbdkR with a short trace and use software to calibrate the feedback timing. Calibrating the feedback delay through software allows the DDR controller to capture the data on the earliest possible Stage 2 sample cycle. Software can delay the feedback pulse by up to half a clock cycle using MCIF0_RFDC[RFFD] . Therefore, as long as the flight time of the MemClkOut0 to data is less than half a clock cycle, there is no reason to match the feedback trace to the round-trip path. When properly calibrated, the feedback pulse is aligned to the first DQS in a 4-beat burst such that the rising edge of DQS is nominally centered on the feedback pulse.

The data captured in Stage 1 is relative to the DQS signals and is sampled again by Stage 2 in order to synchronous the data relative to the DDR 1X clock.

The feedback pulse is sampled in Stage 1 with the data and sampled again in Stage 2. The DDR controller, based on the feedback samples in Stage 2, adjust the Stage 2 sample cycle in order to track variations in timing. See *Read Data Capture Control Register (MCIF0_RDCC)* on page 784 for additional information about Stage 2.

In Stage 3, data is driven onto the PLB bus. The data captured on the rising and falling DQS edges is unpacked into the correct bit locations on the upper (0:63) and lower (64:127) PLB bus.

Figure 22-15. DDR SDRAM Read Circuitry



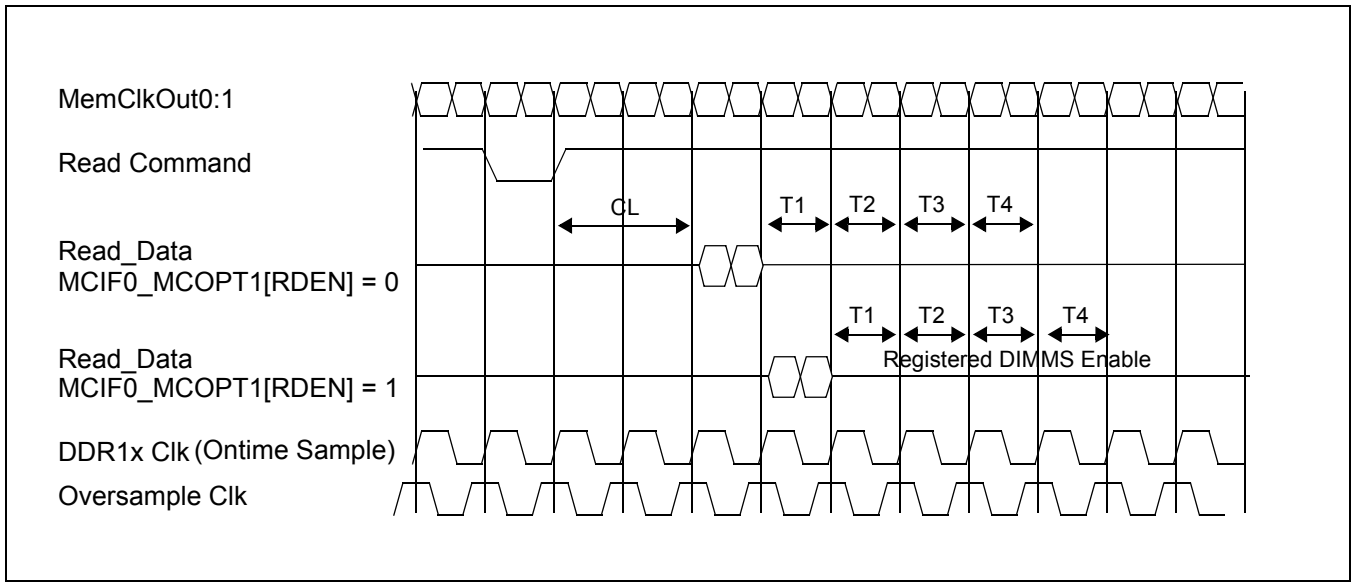
ECC detection and correction if enabled occurs after Stage 3 before completing the read on the PLB.

The following figure shows a timing diagram for a typical read operation. It shows the T1, T2, and T3 cycles, during which Stage 2 captures the data. The feedback control delay, along with oversampling, automatically reduces the total latency by means of the MCIF0_RFDC[RFOS] "guard band." The T1 cycle is the best for capturing the data, but sometimes this cannot be achieved. At T1, T2, or T3, the first two 64-bit groups of data are stored during read sample Stage 2. *Figure 22-17* shows a typical read where data is first sampled in sample cycle T2.

Figure 22-16 shows the on-time and over-sample clocks. These clocks are used by Stage 2 of the read capture logic to sample the feedback signal captured in Stage 1. The feedback signal is sampled by each DQS in Stage 1 with its corresponding byte of data. When the feedback sample in Stage 1 is 1, it qualifies the corresponding byte of data sampled by DQS as data belonging to the first beat of the DDR burst.

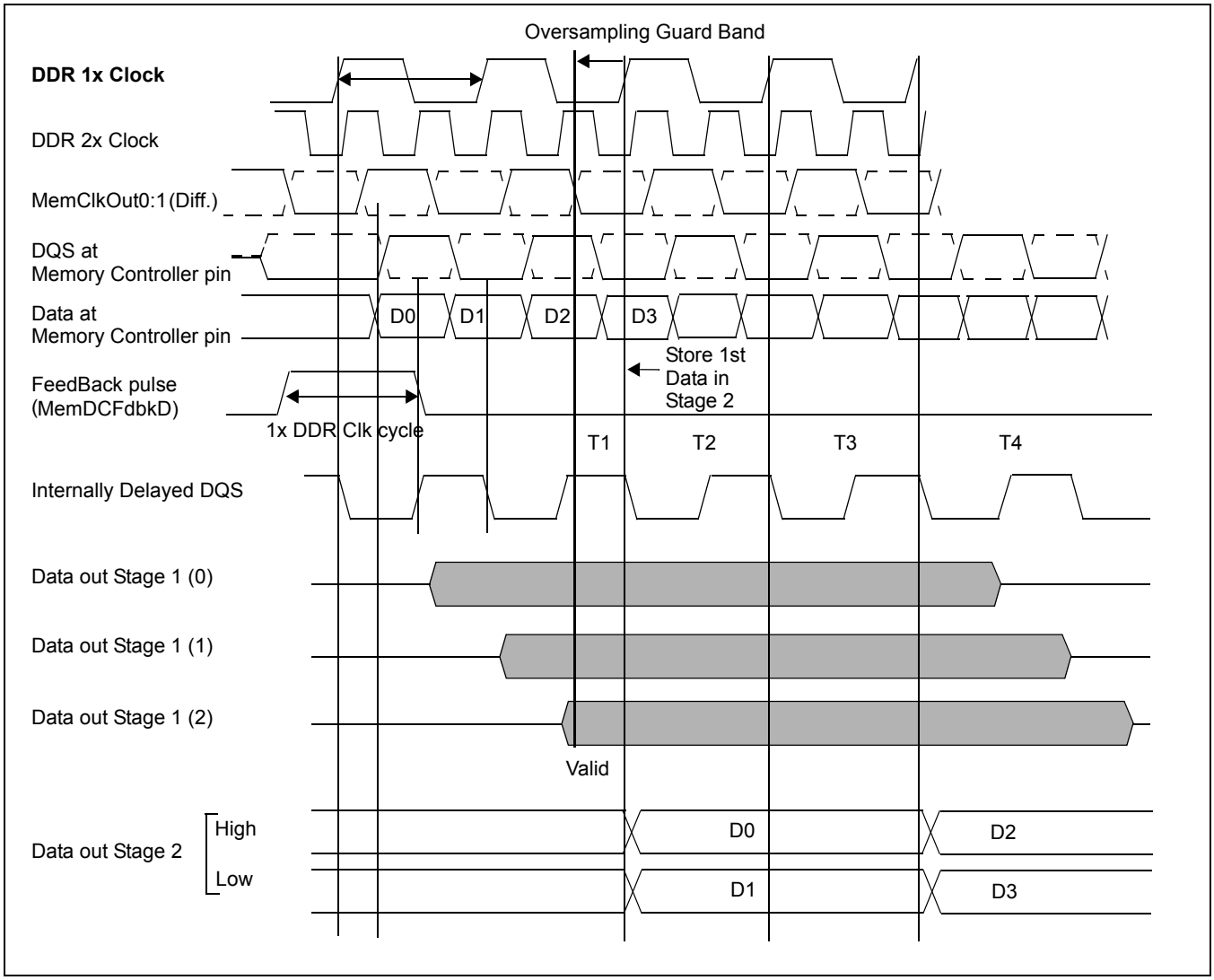
The feedback ontime sample and oversample made by Stage 2 enables the DDR controller to adjust the sample cycle up or down by one cycle as needed between refresh cycles. This adjustment is made depending the logic level of the feedback on-time sample and oversample. Section 23.2.7.16 Read Data Capture Control Register (MCIF0_RDCC) contains a detailed description of the feedback ontime sample and oversample.

Figure 22-16. Read Latency Adjust Timing Diagram



The following figure shows a more detailed example of typical DDR SDRAM read cycle timing.

Figure 22-17. DDR SDRAM Read Cycle Timing



22.2.8 DDR SDRAM Controller Registers

This section describes all DDR SDRAM registers in detail.

22.2.8.1 DDR SDRAM Controller DCR Configuration

After a system reset, software must configure and then enable the DDR SDRAM controller. When this is complete, the memory is ready for access by any master. After the controller is enabled, generally it is not necessary for software to access the SDRAM configuration registers. However, the status registers are useful for determining the state of the memory controller or for querying information about any corrected or uncorrectable ECC errors that might have occurred. All DDR SDRAM controller registers are programmed by means of the DCR bus interface. Accesses to DDR SDRAM controller registers are performed using the indirect addressing method using MCIF0_CFGADDR and MCIF0_CFGDATA.

Table 22-6. DDR SDRAM Controller DCR Addresses

Mnemonic	Register Description	DCR Offset	Access
MCIF0_CFGADDR	Indirect DCR address	0x0010	R/W
MCIF0_CFGDATA	Indirect DCR data	0x0011	R/W

The following code illustrates how to access a DDR SDRAM register by writing the MCIF0_CFGDATA register and then reading back the written value.

```
li r3, register_offset          ! address offset of DDR_SDRAM register
lis r4,<config upper>           ! upper half of configuration data
ori r4, r4,<config lower>       ! lower half of configuration data
mtdcr MCIF0_CFGADDR, r3         ! set offset address
mtdcr MCIF0_CFGDATA, r4         ! write configuration data
mfdcr r5, MCIF0_CFGDATA         ! read back configuration data
```

Reserved fields in the DDR SDRAM controller configuration registers must not change when the register is written. To modify fields within a register, software should read the register, use a mask to clear the target bits, then OR in the new field value, and then write back the result.

22.2.8.2 DDR SDRAM DCR Offset Address Map

The following table provides a complete list of the SDRAM Controller DDR1 and 2 register set defined in the DCR space.

Table 22-7. DDR SDRAM Controller Configuration and Status Registers

Mnemonic	Register	DCR Offset	Access	Reset Value	Page
MCIF0_MCSTAT	Memory Controller Status	0x0014	R	0x6000_0000	765
MCIF0_MCOPT1	Memory Controller Options 1	0x0020	R/W	0x0102_0000	766
MCIF0_MCOPT2	Memory Controller Options 2	0x0021	R/W	0x8400_0000	768
MCIF0_MODT0	On-Die Termination for Rank 0	0x0022	R/W	0x0000_0000	769
MCIF0_MODT1	On-Die Termination for Rank 1	0x0023	R/W	0x0000_0000	769
MCIF0_MODT2	On-Die Termination for Rank 2	0x0024	R/W	0x0000_0000	769
MCIF0_MODT3	On-Die Termination for Rank 3	0x0025	R/W	0x0000_0000	769
MCIF0_CODT	On-Die Termination for Controller	0x0026	R/W	0x0000_0000	772
MCIF0_RTR	Refresh Timer Register	0x0030	R/W	0x0BE0_0000	774
MCIF0_MB0CF	Memory Rank 0 Configuration	0x0040	R/W	0x0000_0000	776
MCIF0_MB1CF	Memory Rank 1 Configuration	0x0044	R/W	0x0000_0000	776
MCIF0_MB2CF	Memory Rank 2 Configuration	0x0048	R/W	0x0000_0000	776
MCIF0_MB3CF	Memory Rank 3 Configuration	0x004C	R/W	0x0000_0000	776

Table 22-7. DDR SDRAM Controller Configuration and Status Registers (Continued)

Mnemonic	Register	DCR Offset	Access	Reset Value	Page
MCIF0_INITPLR0	Initialization Preload Register 0	0x0050	R/W	0x0000_0000	776
MCIF0_INITPLR1	Initialization Preload Register 1	0x0051	R/W	0x0000_0000	776
MCIF0_INITPLR2	Initialization Preload Register 2	0x0052	R/W	0x0000_0000	776
MCIF0_INITPLR3	Initialization Preload Register 3	0x0053	R/W	0x0000_0000	776
MCIF0_INITPLR4	Initialization Preload Register 4	0x0054	R/W	0x0000_0000	776
MCIF0_INITPLR5	Initialization Preload Register 5	0x0055	R/W	0x0000_0000	776
MCIF0_INITPLR6	Initialization Preload Register 6	0x0056	R/W	0x0000_0000	776
MCIF0_INITPLR7	Initialization Preload Register 7	0x0057	R/W	0x0000_0000	776
MCIF0_INITPLR8	Initialization Preload Register 8	0x0058	R/W	0x0000_0000	776
MCIF0_INITPLR9	Initialization Preload Register 9	0x0059	R/W	0x0000_0000	776
MCIF0_INITPLR10	Initialization Preload Register 10	0x005A	R/W	0x0000_0000	776
MCIF0_INITPLR11	Initialization Preload Register 11	0x005B	R/W	0x0000_0000	776
MCIF0_INITPLR12	Initialization Preload Register 12	0x005C	R/W	0x0000_0000	776
MCIF0_INITPLR13	Initialization Preload Register 13	0x005D	R/W	0x0000_0000	776
MCIF0_INITPLR14	Initialization Preload Register 14	0x005E	R/W	0x0000_0000	776
MCIF0_INITPLR15	Initialization Preload Register 15	0x005F	R/W	0x0000_0000	776
MCIF0_RQDC	Read DQS Delay Control Register	0x0070	R/W	0x0000_0000	782
MCIF0_RFDC	Read Feedback Delay Control Register	0x0074	R/W	0x0000_0000	782
MCIF0_RDCC	Read Data Capture Control Register	0x0078	R/W	0x0000_0000	784
MCIF0_DLCR	Delay Line Calibration Register	0x007A	R/W	autoinit status	785
MCIF0_CLKTR	Memory Clock Timing Register	0x0080	R/W	0x0000_0000	786
MCIF0_WRDTR	Write Data/DM/DQS Timing Register	0x0081	R/W	0x8600_0800	786
MCIF0_SDTR1	SDRAM Timing Register 1	0x0085	R/W	0x8020_1000	787
MCIF0_SDTR2	SDRAM Timing Register 2	0x0086	R/W	0x4210_4442	788
MCIF0_SDTR3	SDRAM Timing Register 3	0x0087	R/W	0x1010_0C10	789
MCIF0_MMODE	SDRAM Mode Register	0x0088	R/W	0x0000_0042	789
MCIF0_MEMODE	SDRAM Extended Mode Register	0x0089	R/W	0x0000_0400	790
MCIF0_ECCES	ECC Error Status Register	0x0098	R/W	0x0000_0000	791
MCIF0_FCSR	Feedback Calibration Status Register	0x00B0	RO	0x0000_0000	792
MCIF0_RTSR	Run Time Tracking Status Register	0x00B1	RO	0xFFFF_F000	792

22.2.8.3 DCR Modifications After Enabling SDRAM Memory Controller

The registers described in *Table 22-8* apply globally to the memory subsystem and are used to configure its operation. Read/Write access is restricted by hardware to certain registers once the SDRAM Controller is enabled because MCIF0_MCOPT2[DCEN] = 1.

Table 22-8 lists the complete set of configuration registers and the effects on each register of reading and writing to it with MCIF0_MCOPT2[DCEN] asserted/deasserted.

Note: Reserved fields in the memory configuration registers must not change when the register is written. Changing reserved fields can cause DDR SDRAM controller malfunction

Table 22-8. Device Configuration Register Dependence on MCIF0_MCOPT2[DCEN]

Register Name	Description	DCR Offset	MCIF0_MCOPT2[DCEN] See Note	Transaction	Result
MCIF0_MCSTAT	Memory Controller Status	0x0014	X	W	Update
MCIF0_MCOPT1	Memory Controller Options 1	0x0020	X	W	Update
MCIF0_MCOPT2	Memory Controller Options 2	0x0021	X	W	Update
MCIF0_MODT0-3	On-Die Termination for Ranks 0-3	0x0022-25	X	W	Update
MCIF0_CODT	On-Die Termination for Controller	0x0026	X	W	Update
MCIF0_RTR	Refresh Timer Register	0x0030	X	W	Update
MCIF0_MB0CF	Memory Rank 0 Configuration	0x0040	0	W	Update
			1	W	No Change
MCIF0_MB1CF	Memory Rank 1 Configuration	0x0044	0	W	Update
			1	W	No Change
MCIF0_MB2CF	Memory Rank 2 Configuration	0x0048	0	W	Update
			1	W	No Change
MCIF0_MB3CF	Memory Rank 3 Configuration	0x004C	0	W	Update
			1	W	No Change
MCIF0_INITPLR[0:15]	Manual Initialization Control Register [0:15]	0x0050-5F	X	W	Update
MCIF0_RQDC	Read DQS Delay Control Register	0x0070	X	W	Update
MCIF0_RFDC	Read Feedback Delay Control Register	0x0074	X	W	Update
MCIF0_RDCC	Read Data Capture Control Register	0x0078	X	W	Update
MCIF0_DLYCAL	Delay Line Calibration Register	0x007A	X	W	Update
MCIF0_CLKTR	Memory Clock Timing Register	0x0080	X	W*	Update
MCIF0_WRDTR	Write Data/DM/DQS Clock Timing Register	0x0081	X	W*	Update
MCIF0_SDTR1-3	SDRAM Timing Registers 1-3	0x0085-87	X	W*	Update
MCIF0_MMODE	SDRAM Mode Register	0x0088	X	W*	Update
MCIF0_MEMODE	SDRAM Extended Mode Register	0x0089	X	W*	Update

Table 22-8. Device Configuration Register Dependence on MCIF0_MCOPT2[DCEN] (Continued)

Register Name	Description	DCR Offset	MCIF0_MCOPT2[DCEN] See Note	Transac-tion	Result
MCIF0_ECCES	ECC Error Status Register	0x0098	X	W	No Change
any	any	any	X	R	Read Data
Note: An X in the MCIF0_MCOPT2[DCEN] column indicates do not care.					

The logic allows modification of MCIF0_SDTR1–3, MMODE, MEMODE, CLKTR, or WRDTR after enabling because MCIF0_MCOPT2[DCEN] = 1. However, it is not recommended to change MCIF0_SDTR1–3, MCIF0_MMODE, and MCIF0_MEMODE after enabling the DDR controller.

Also, when changing MCIF0_CLKTR, the downstream effects (such as minimum lock time, and re-establishing lock) on the system/chip level PLL(s) must be considered before attempting to access the memory. Modifying MCIF0_CLKTR after setting MCIF0_MCOPT2[DCEN] is not recommended.

22.2.8.4 Initial Configuration Following Power-On Reset

All SDRAM-specific registers must be configured before enabling the DDR SDRAM controller, according to the SDRAM initialization. MCIF0_SDTR1–3, MCIF0_MMODE, MCIF0_MEMODE, MCIF0_RTR, and all MCIF0_MBnCF registers must be configured before setting MCIF0_MCOPT2[DCEN] = 1.

Write access to the DDR SDRAM controller error status registers (MCIF0_ECCES). is independent of the MCIF0_MCOPT2 value. All DCR writes to the error status registers are given the highest priority in case a DCR error status register write occurs at the same time as an error status register update.

DCR reads complete normally, with the contents of the targeted register returned independent of the MCIF0_MCOPT2[SREN] value.

22.2.8.5 Re-Configuration Following Initial Configuration

Reconfiguration of the SDRAM-specific registers can be performed at any time after initial configuration. Before reconfiguring, all pending and queued requests targeting the SDRAM controller must be allowed to complete. The following list shows the sequence required:

1. Ensure that all pending and queued memory requests are complete.
2. Disable the DDR SDRAM controller by writing MCIF0_MCOPT2[DCEN] = 0.
3. Update SDRAM-specific timing registers.
4. If any changes are made to the MCIF0_MEMODE or MCIF0_MMODE registers, then the memory devices must be reinitialized so that these changes are passed to the memory devices. This must be done by means of the MCIF0_INITPLRn registers, in a manner recommended by the memory device manufacturer. For example, this may involve issuing Extended Mode Register and Mode Register Set commands, and the reset of the memory DLL.
5. Enable the DDR SDRAM controller - Write MCIF0_MCOPT2[DCEN] = 1.

The memory can now be used.

22.2.8.6 Memory Controller Status Register (MCIF0_MCSTAT)

MCIF0_MCSTAT provides real-time status to the system software about the operation of the DDR SDRAM controller. The information returned indicates the controller status at the time of the DCR read.

Figure 22-18. Memory Controller Status Register (MCIF0_MCSTAT)

0	MIC	Memory Initialization Complete: 0 Memory Initialization Not Complete 1 Memory Initialization Complete	Indicates the status of memory initialization: This bit is set to 1 to indicate that the memory subsystem is ready for use. This occurs when the preloaded initialization sequence has completed to the memory devices. The bit is cleared whenever the Initialization Preload Trigger register bit (IPTR) in MCIF0_MCOPT2 is set to one.
1	SRMS	Self-Refresh Mode Status: 0 SDRAM is not in Self-Refresh Mode 1 SDRAM is in Self-Refresh Mode	This bit is set upon the successful completion of Self-Refresh Mode entry. That is, the SDRAM device has been placed in Self-Refresh Mode. This bit is cleared when Self-Refresh Mode has been exited. This bit applies to both software-initiated self-refresh mode by means of MCIF0_MCOPT2[SREN], and to hardware-initiated self-refresh mode by means of the assertion of the HISRRst signal.
2	IDLE	Controller Idle Status: 0 Busy 1 Idle	Indicates that there are no current or pending queued read/write accesses.
3:31		Reserved	

22.2.8.7 Memory Controller Options 1 Register (MCIF0_MCOPT1)

MCIF0_MCOPT1 enables the configuration of the DDR SDRAM interface controller-specific options. See the related DDR SDRAM sections of this document for specific details and resultant processing. All application-specific options selected by means of MCIF0_MCOPT1 and MCOPT2, and MCIF0_SDTR1–3 must be configured before enabling the SDRAM Controller. The following figure describes the MCIF0_MCOPT1 register bits.

Note: MCIF0_MCOPT1[MCHK] can be changed from (0b01, 0b10 or 0b11) to (0b01, 0b10 or 0b11) but it is not recommended to change from 0b00 to any other configuration. The ECC calculation is made in series with a read and affects read timing.

Figure 22-19. Memory Controller Options 1 Register (MCIF0_MCOPT1)

0:1		Reserved	
2:3	MCHK	Memory Data Error Checking: 00 None 01 ECC checking and correction with no error reporting. 10 ECC generation only (no checking or correction) 11 ECC checking and correction with error reporting.	
4	RDEN	Registered DIMM Enable: 0 Disable 1 Enable	Add one clock cycle as required for registered memory: See *Figure 22-16* on page 759 for details.
5:6	PMUM	Page Management Unit Mode: 00 Close: Always close pages after a read or write command. 01 AutoClose: Close pages after a read or write unless a request for that page is queued. 10 Open: After use, pages will remain open until a read or write to a different page is queued, or a memory refresh is done. 11 Reserved	
7	WDTH	DDR DRAM Data Width: 0 32-bit 1 64-bit	
8:9	UIOS	Unused IO State: 00 Active (Hi-Z on Read, Driven on Write) - switching DQS 01 Active (Hi-Z on Read, Driven on Write) - static values 10 Static (Always Driven) - static values 11 Tri-stated	This field can be used to control the state, by means of the corresponding IO driver data and enable signals of the unused IO for various configurations. Unused IO are defined as the IO associated with a function that is not enabled. For example, ECC[0:7], DM[8], and DQS[8] are unused IO if ECC is not enabled. Similarly, MemData[32:63], DM[4:7], and DQS[4:7] are unused IO in 32-bit mode. All unused IO receivers are always gated off.
10	BCNT	Memory Device Bank Count: 0 4 Bank Devices 1 8 Bank Devices	
11	DDR2	DDR2/DDR1 Select: 0 DDR1 Memory Type 1 DDR2 Memory Type	Select I/O voltage DDR1 2.5V DDR2 1.8V
12:13		Reserved	

14:15	QDEP	Depth of Command Queue for Page Lookahead/Reordering: 00 Reserved 01 Reserved 10 4 Cmds Deep 11 Reserved	
16	RWOO	Enable Out of Order Reads and Writes by the Controller: 0 Disabled 1 Enabled	Out of Order must be enabled by all controls to permit out of order operations.
17	WOOO	Enable Out of Order Writes by the Controller: 0 Disabled. 1 Enabled.	
18	DCOO	Disable All Out of Order Operations: 0 Enable page management out of order 1 Disable page management out of order	Disables all out of order operations, including page management
19	DREF	Enable Deferred Refresh: 0 Normal Refresh 1 Defer up to 4 refresh commands if the controller is busy.	This function modifies the auto refresh timing to avoid delaying queued read and write commands:
20:31		Reserved	

Effect of Unused I/Os (UIOS)

The UIOS settings enable the user to configure the processing of the unused IO (if any) at the chip level.

There are two potential unused IO groupings:

1.) ECC0:7, DM8, and DQS8 for ECC disabled.

2.)MemData32:63, DM4:7, and DQS4:7 for 32-bit mode enabled.

The processing of these IO groupings, based on the selected UIOS setting, is shown in *Table 22-9* and *Table 22-10*.

Table 22-9. Unused IO - ECC Disabled

Unused IO	UIOS			
	00	01	10	11
	Write/Read	Write/Read	Write/Read	Write/Read
DM8	1/1	1/1	1/1	1/1
DQS8	T/Hi-Z	0/Hi-Z	0/0	Hi-Z/Hi-Z
ECC0:7	0's/Hi-Z	0's/Hi-Z	0's/0's	Hi-Z/Hi-Z

Table 22-10. Unused IO - 32-bit Mode Enabled

Unused IO	UIOS			
	00	01	10	11
	Write/Read	Write/Read	Write/Read	Write/Read
DM4:7	1's/1's	1's/1's	1's/1's	1's/1's
DQS4:7	T/Hi-Z	0's/Hi-Z	0's/0's	Hi-Z/Hi-Z
MemData32:63	0's/Hi-Z	0's/Hi-Z	0's/0's	Hi-Z/Hi-Z
Note: T: Toggles on Write Cycles Hi-Z: High Impedance				

22.2.8.8 Memory Controller Options 2 Register (MCIF0_MCOPT2)

MCIF0_MCOPT2 enables the configuration of the DDR SDRAM controller, initialization sequence, and self-refresh features. See the related DDR SDRAM sections of this document for specific details and resultant processing. The following figure describes the MCIF0_MCOPT2 register bits.

Figure 22-20. Memory Controller Options 2 Register (MCIF0_MCOPT2)

0	SREN	Self Refresh Entry/Exit Control: 0 Exit self refresh mode. 1 Enter self refresh mode.	This bit is used to control self-refresh mode entry and exit. It is set to 1 at power on reset, and may also be set to 1 by input signal HISRRst. Software may also set this bit to force self-refresh mode entry. Setting this bit by means of either software or hardware causes DCEN (bit 4) of this register to be cleared to zero, disabling the controller and halting auto refresh. Clearing this bit causes self-refresh mode exit by re-enabling ClkEn to the memories. Software must then perform any memory device initialization before setting DCEN and allowing normal memory operations. If XSRP (bit 3) is set, self refresh exit is prevented.
1		Reserved	
2	IPTR	Initialization Trigger Register: 0 Initialization Sequencer Idle 1 Execute Preloaded Initialization Sequence / Initialization Sequencer Busy	Writing a 1 to IPTR triggers the preloaded initialization sequence (as configured in the MCIF0_INITPLRn registers) to be initiated to the memory devices. MCIF0_MCSTAT[MIC] is automatically cleared (1'b0) as a result. If set to 1, this bit remains set until hardware completes the initialization sequence. After the preloaded initialization sequence is complete, this bit is cleared and MCIF0_MCSTAT[MIC] is set to 1. This bit should never be set to 0 by software. It must only be written to a 1 by software if, and only if, the initialization sequencer is Idle (IPTR = 0).
3	XSRP	Exit Self Refresh Prevent: 0 Allow self refresh exit in all conditions. 1 Prevent the controller and memory from exiting self refresh mode except for the first exit following a power down/up cycle or a reset.	This bit must be set and cleared by software. Once this bit is set, the memory may enter self refresh, but cannot exit self refresh until this bit is cleared by software or a reset.

4	DCEN	SDRAM Controller Enable: 0 Controller Disabled 1 Controller Enabled for normal operation.	This bit allows software to enable or disable normal memory operations and auto refresh. Before setting this bit true, software must exit self refresh (if enabled), program all memory timing and control registers to valid values, and perform the correct memory initialization sequence as specified by the device manufacturer. Clearing this bit prevents memory commands from being accepted, and disables auto refresh. This bit is cleared by reset or by entry into self-refresh mode.
5	ISIE	Initialization Sequence Interruptible: 0 Disabled 1 Enabled	This bit enables the preloaded initialization sequence to be interrupted if a hardware-initiated self-refresh request is asserted during the initialization sequence. If disabled, the preloaded initialization sequence runs to completion before the hardware-initiated self-refresh request is acted upon.
6:31		Reserved	

22.2.8.9 Memory On-Die Termination Control Register (MCIF0_MODTn)

DDR2 memory devices optionally support On-Die Termination (ODT). Switchable ODT is available on DDR2 Memory device I/O signals and the PPC460EX/EXr/GT's memory controller I/O signals. This enables dynamic control of termination to improve signal quality and to reduce idle power consumption.

Termination resistance (which is internal to the DDR2 SDRAM device) can be enabled by using ODT signal outputs MemODT0:3 from the memory controller. These are controlled by the MCIF0_MODT*n* registers (where *n* = rank number).

The PPC460EX/EXr/GT supports four ranks (using signals $\overline{\text{BankSel0:3}}$) and ODT inside each of these ranks can be turned on and off during read or write operations of Rank *n* itself or another Rank by the settings in the MCIF0_MODT*n* register.

The MCIF0_MODTn register controls the MemODTn signals with n = 0–3.
For example, the MCIF0_MODT0 register controls the MemODT0 signal which turns on and off the ODT for the DRAM Device connected to Rank 0 ($\overline{\text{BankSel0}}$) signal.

Figure 22-21. Memory On-Die Termination Control Register (MCIF0_MODTn)

0	EOA	MemODTn Always 0 Disable 1 Enable	When EOA = 1, the memory's ODT is enabled for reads and writes.
1	EB3W	ODTn with Rank 3 Write 0 Disable 1 Enable	
2	EB3R	ODTn with Rank 3 Read 0 Disable 1 Enable	
3	EB2W	ODTn with Rank 2 Write 0 Disable 1 Enable	
4	EB2R	ODTn with Rank 2 Read 0 Disable 1 Enable	

5	EB1W	ODTn with Rank 1 Write 0 Disable 1 Enable	
6	EB1R	ODTn with Rank 1 Read 0 Disable 1 Enable	
7	EB0W	ODTn with Rank 0 Write 0 Disable 1 Enable	
8	EB0R	ODTn with Rank 0 Read 0 Disable 1 Enable	
9:31		Reserved	
Note: The ODT drive control pins can be set to hi-Z by the MCIF0_CODT[ODTS] bit.			

Example:
Table 22-11 provides typical MCIF0_MODTn register settings for different combinations of single and double-sided DIMMs. *Table 22-12* and *Table 22-13* indicate when the ODT signals are driven for read and write accesses and includes recommended memory termination for each configuration. These tables are only an example and may not be the best configuration for all implementations. Some experimentation may be needed for a specific board design.

Table 22-11. ODT0:3 Signal Configuration for DDR2 DIMMs (Only ODT0 and ODT2 Driven)

DIMMs Installed	$\overline{\text{BankSel0}}$	$\overline{\text{BankSel1}}$	$\overline{\text{BankSel2}}$	$\overline{\text{BankSel3}}$	M0DT0	M0DT1	M0DT2	M0DT3
2R/2R	P	P	P	P	0x78000000	0x00000000	0x07800000	0x00000000
2R/1R	P	P	P	U	0x18000000	0x00000000	0x07800000	0x00000000
1R/2R	P	U	P	P	0x78000000	0x00000000	0x01800000	0x00000000
1R/1R	P	U	P	U	0x18000000	0x00000000	0x01800000	0x00000000
2R/Empty	P	P	U	U	0x05000000	0x00000000	0x00000000	0x00000000
Empty/2R	U	U	P	P	0x00000000	0x00000000	0x50000000	0x00000000
1R/Empty	P	U	U	U	0x01000000	0x00000000	0x00000000	0x00000000
Empty/1R	U	U	P	U	0x00000000	0x00000000	0x10000000	0x00000000

Notes:
1. 2R = double rank DIMM, 1R = single rank DIMM
2. U = unpopulated, P = populated
3. MOD0 = MCIF0_MOD0, MOD1 = MCIF0_MOD1, MOD2 = MCIF0_MOD2, MOD3 = MCIF0_MOD3
4. Settings enable MemODT0:3 when writing to DDR2 memory.
5. For DDR1, set MCIF0_MOD0 = MCIF0_MOD1 = MCIF0_MOD2 = MCIF0_MOD3 = 0x00000000.

Table 22-12. ODT Signals Driven on a Read Access using Table 22-11 Settings

DIMMs Installed	Rank Accessed on Read	0DT0	0DT1	0DT2	0DT3	Memory Termination in EMR1[1]
2R/2R	Rank 0	Driven	-	-	-	50 or 75 Ohm
	Rank 1	Driven	-	-	-	
	Rank 2	-	-	Driven	-	
	Rank 3	-	-	Driven	-	
2R/1R	Rank 0	Driven	-	-	-	50 or 75 Ohm
	Rank 1	Driven	-	-	-	
	Rank 2	-	-	Driven	-	
1R/2R	Rank 0	Driven	-	-	-	50 or 75 Ohm
	Rank 2	-	-	Driven	-	
	Rank3	-	-	Driven	-	
1R/1R	Rank 0	Driven	-	-	-	50 or 75 Ohm
	Rank 2	-	-	Driven	-	
2R/Empty	Rank 0	Driven	-	-	-	150 Ohm
	Rank 1	Driven	-	-	-	
Empty/2R	Rank 2	-	-	Driven	-	150 Ohm
	Rank 3	-	-	Driven	-	
1R/Empty	Rank 0	Driven	-	-	-	150 Ohm
Empty/1R	Rank 2	-	-	Driven	-	150 Ohm

Note 1. Memory termination in EMR1 (Extended Mode Register 1) configured in MCIF0_INITPLLn and MCIF0_MMODE.

Table 22-13. ODT Signals Driven on a Write Access using Table 22-11 Settings

DIMMs Installed	Rank Accessed on Write	0DT0	0DT1	0DT2	0DT3	Memory Termination in EMR1[1]
2R/2R	Rank 0	-	-	Driven	-	50 or 75 Ohm
	Rank 1	-	-	Driven	-	
	Rank 2	Driven	-	-	-	
	Rank 3	Driven	-	-	-	
2R/1R	Rank 0	-	-	Driven	-	50 or 75 Ohm
	Rank 1	-	-	Driven	-	
	Rank 2	Driven	-	-	-	

Table 22-13. ODT Signals Driven on a Write Access using Table 22-11 Settings (Continued)

DIMMs Installed	Rank Accessed on Write	0DT0	0DT1	0DT2	0DT3	Memory Termination in EMR1[1]
1R/2R	Rank 0	-	-	Driven	-	50 or 75 Ohm
	Rank 2	Driven	-	-	-	
	Rank3	Driven	-	-	-	
1R/1R	Rank 0	-	-	Driven	-	50 or 75 Ohm
	Rank 2	Driven	-	-	-	
2R/Empty	Rank 0	Driven	-	-	-	150 Ohm
	Rank 1	Driven	-	-	-	
Empty/2R	Rank 2	-	-	Driven	-	150 Ohm
	Rank 3	-	-	Driven	-	
1R/Empty	Rank 0	Driven	-	-	-	150 Ohm
Empty/1R	Rank 2	-	-	Driven	-	150 Ohm

Note 1. Memory termination in EMR1 (Extended Mode Register 1) configured in MCIF0_INITPLLn and MCIF0_MMODE.

22.2.8.10 Controller ODT and I/O Pin Control Register (MCIF0_CODT)

MCIF0_CODT controls the operation of the memory controller on-die-termination networks associated with the data handling signals and groups of control signals.

Dynamic on-die termination is available on both the DDR2 memory I/O device signals and the memory controller's Data, DQS and DM signals. This allows selective enablement of ODT during read vs. write cycles to improve signal quality and to reduce idle power consumption. (Note that bits 0:8 control only the Controller's Data, DQS and DM output signals and have no effect on the remaining signals.)

In addition, static control of the remaining signal's termination is available for the following groups of control signals:

- (ClkEn0:3, BankSel0:3 and MemODT0:3)
- (WE, CAS, RAS, and MemAddr14:0)
- (MemDCFdbkD)
- (MemDCFdbkR)

Figure 22-22. Controller ODT and I/O Pin Control Register (MCIF0_CODT)

0	EOA	Controller ODT Always 0 Disable 1 Enable	When EOA = 1, the DDR controller's internal ODT is enabled for reads and writes.
1	EB3W	Controller ODT with Rank 3 Write 0 Disable 1 Enable	
2	EB3R	Controller ODT with Rank 3 Read 0 Disable 1 Enable	

3	EB2W	Controller ODT with Rank 2 Write 0 Disable 1 Enable	
4	EB2R	Controller ODT with Rank 2 Read 0 Disable 1 Enable	
5	EB1W	Controller ODT with Rank 1 Write 0 Disable 1 Enable	
6	EB1R	Controller ODT with Rank 1 Read 0 Disable 1 Enable	
7	EB0W	Controller ODT with Rank 0 Write 0 Disable 1 Enable	
8	EB0R	Controller ODT with Rank 0 Read 0 Disable 1 Enable	
9:10	ODTSH	Controller ODT Timing 00 Normal controller ODT timing 01 Remove one cycle at end of ODT enable 10 Add one cycle to beginning of ODT enable 11 Shift ODT timing one cycle earlier	
11	CODTZ	Controller ODT resistance 0 75 ohms 1 Reserved	
12	CKEG	Controller ODT termination on ClkEn, BankSel, and MemODT pins 0 Disable (typical) 1 Enable	
13	CTLG	Controller ODT termination on $\overline{WE}$, $\overline{RAS}$, $\overline{CAS}$, and Address pins 0 Disable (typical) 1 Enable	
14	FBDG	Controller ODT termination on MemDCFdbkD pin 0 Disable (typical) 1 Enable	
15	FBRG	Controller ODT termination on MemDCFdbkR pin 0 Disable (typical) 1 Enable	
16		Reserved	
17	DQLZ	Memory DQS driver impedance mode 0 36Ω (reduced strength driver) 1 18Ω (full strength driver)	The memory clock and DQS signals are the only DDR drivers with a programmable output impedance. DDR data, address, control drivers are reduced strength drivers (36Ω output impedance).
18	CKLZ	MemClkOut driver impedance mode 0 36Ω (reduced strength driver) 1 18Ω full strength driver)	
19:25		Reserved	
26	2518	Memory mode I/O 0 2.5V mode (DDR1) 1 1.8Vmode (DDR2)	

27	DQSE	DQS I/O 0 Differential mode 1 Single-end mode	
28	CKSE	Memory clock (MemClkOut) 0 Differential mode 1 Single-end mode	
29:30		Reserved	
31	ODTS	MemODT driver mode 0 Hi-Z 1 Normal	

Table 22-14 provides typical MCIF0_CODT register settings for different combinations of single and double-sided DIMMs. This table is only a recommendation. Some experimentation may be needed for a specific board design.

Table 22-14. DDR Memory Controller ODT Configuration for DDR2 DIMMs

Ranks Installed	$\overline{\text{BankSel0}}$	$\overline{\text{BankSel1}}$	$\overline{\text{BankSel2}}$	$\overline{\text{BankSel3}}$	MCIF0_CODT
2R/2R	P	P	P	P	0x2A800021
2R/1R	P	P	P	U	0x0A800021
1R/2R	P	U	P	P	0x28800021
1R/1R	P	U	P	U	0x08800021
2R/Empty	P	P	U	U	0x02800021
Empty/2R	U	U	P	P	0x28000021
1R/Empty	P	U	U	U	0x00800021
Empty/1R	U	U	P	U	0x08000021

Notes:
1. 2R = double rank DIMM, 1R = single rank DIMM
2. U = unpopulated, P = populated
3. Settings enable the following:
   - Enables 1.8V I/O.
   - Enable differential signaling for DQS and MemClkOut.
   - Configures MemClkOut and DQS signals for low impedance (reduced strength drive).
   - Enables DDR controller internal 75 ohm ODT for reads.
4. For DDR1, disable ODT, select DDR1 I/O mode, set DQS drive strength and set MemClkOut drive strength.

22.2.8.11 Refresh Timer Register (MCIF0_RTR)

The 16-bit field of MCIF0_RTR (bits 0–15) determines the memory refresh rate for the SDRAM. The internal counter runs at the controller clock frequency. Thus, at 200 MHz, a value of 0x0BE0 produces a refresh interval of 15.20μs (3040 times 5ns equals 15.20μs). This register must be programmed to accommodate the DDR SDRAM device-specific refresh interval for the target operating frequency. Note that bit 0 is the MSB and bit 15 is the LSB.

Bits 13–15 are always 0, to reduce logic complexity.

Deferred refresh is enabled by setting MCIF0_MCOPT1[DREF] (bit 19). If this bit is set, refresh commands are deferred until either the controller command queue goes empty or the number of deferred refresh commands equals four. If either of these conditions is reached, the controller issues the deferred refresh commands without intervening gaps.

Normal refresh is recommended if normal memory operations cannot be delayed longer than the programmed refresh period. The following figure describes the MCIF0_RTR register bits.

Figure 22-23. Refresh Timer Register (MCIF0_RTR)			
0:12	RTIN	Refresh Timing Interval	
13:15		Hardcoded to 0	
16:31		Reserved	

Table 22-15. MCIF0_RTR Settings for Various tREF and MemClkOut Values

tREF(μs)	MemClkOut (MHz)	MCIF0_RTR
7.8	133.33	0x04100000
	166.66	0x05100000
	177.77	0x05680000
	200.00	0x06180000
	233.33	0x07180000
	250.00	0x07980000
	266.66	0x08200000
15.2	133.33	0x07E80000
	166.66	0x09E00000
	177.77	0x0A880000
	200.00	0x0BE00000
	233.33	0x0DD80000
	250.00	0x0ED80000
	266.66	0x0FD00000

22.2.8.12 Memory Rank 0-3 Configuration Register (MCIF0_MBnCF)

The MCIF0_MBnCF registers must be configured to define the addressing mode—row (RAS), column (CAS), and bank address (BA) associated with each memory Rank.

The base address and size of a Rank are defined in the MQ0_BnBAS registers. (See *Rank n Base Address and Size Register (MQ0_BnBAS)* on page 744 for details.) The following figure describes the MCIF0_MBnCF register bits.

Figure 22-24. Memory Rank 0-3 Configuration Register (MCIF0_MBnCF)

0:19		Reserved	
20:23	M_AM	Addressing Mode 0000 Mode 0 0001 Mode 1 0010 Mode 2 0011 Mode 3 0100 Mode 4 0101 Mode 5 0110 Mode 6 0111 Mode 7 1000 Mode 8 1001 Mode 9 Others - Reserved	SDRAM Memory Organization Row (RAS) / Column (CAS) / Logical Banks (BA) Mode 0: N x 8 4 Bank Mode 1: N x 9 4 Bank Mode 2: N x 10 4 Bank Mode 3: N x 11 4 Bank Mode 4: N x 12 4 Bank Mode 5: N x 8 8 Bank Mode 6: N x 9 8 Bank Mode 7: N x 10 8 Bank Mode 8: N x 11 8 Bank Mode 9: N x 12 8 Bank
24:30		Reserved	
31	M_BE	Memory Rank Enable	

22.2.8.13 Initialization Preload Register (MCIF0_INITPLRn)

This group of 16 registers provides software with a way to configure the various steps of the SDRAM initialization sequence. The initialization sequence of the SDRAM device is fully programmable. The Enable bit (IENA) must be set for each entry in the sequence to be executed. The following figure describes the MCIF0_INITPLRn register bits.

Note: The Mode Register Set and Extended Mode Register Set commands issued during the initialization sequence must configure the SDRAM in a mode consistent with the configuration of the controller as defined in the SDRAM Mode register (MCIF0_MMODE) located at 0x88 and the SDRAM Extended Mode register (MCIF0_MEMODE) located at 0x89, and comply with the restrictions defined there.

Figure 22-25. Initialization Preload Register (MCIF0_INITPLRn)

0	IENA	Enable	Enables direct software access to the SDRAMs.
1:8	IMWT	Minimum Wait	Determines the minimum time before any subsequent command may be issued to the memory device following the currently executing command.
9		Reserved	
10:12	ICMD	Initialization Command $\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$	This 3-bit field maps directly to $\overline{RAS}$, $\overline{CAS}$, and $\overline{WE}$, respectively, and defines the command issued in conjunction with the assertion of chip select to the memory devices. A 1 value corresponds to the signal being driven high; a 0 corresponds to the signal being driven low.

13:15	IBA	Initialization Bank Address BA[2:0]	This 3-bit field maps directly to BA[2:0], respectively, and defines the bank address issued in conjunction with the assertion of chip select to the memory devices.
16		Reserved	
17:31	IMA	Initialization Memory Address MemAddr14:0	This 15-bit field maps directly to MemAddr14:0. See the JEDEC standard for the definition of the command fields used in DDR SDRAM Mode Register Set (MRS) and Extended MRS (EMRS). The command is issued in conjunction with the assertion of chip select to the memory devices.

When exiting reset, the memory is disabled, with ClkEn0:3 held low. Before initialization, the CKE pin must go high, which is done by clearing the SREN bit of the MCIF0_MCOPT2 register. Similarly, if initialization is required, the initialization sequence must be triggered, by setting the IPTR bit in the MCIF0_MCOPT2 register.

A typical initialization procedure for DDR1 operating at 133MHz, CAS Latency = 3, Burst Length = 4, Burst Type = sequential, is as follows:

dcr_write conf_adr 0x0050 // INITPLR0

dcr_write conf_dta 0x81B80000 // no-op; wait min 3 cycles

dcr_write conf_adr 0x0051 // INITPLR1

dcr_write conf_dta 0x83900400 // precharge all; wait min 7 cycles (set wait to tRP)

dcr_write conf_adr 0x0052 // INITPLR2

dcr_write conf_dta 0x81810000 // EMRS - enable DLL; wait min 3 cycles (set wait to tMRD)(see **Note**)

dcr_write conf_adr 0x0053 // INITPLR3

dcr_write conf_dta 0xFF800132 // MRS - reset DLL; wait 255 cycles (set wait to tMRD)(see **Note**)

dcr_write conf_adr 0x0054 // INITPLR4

dcr_write conf_dta 0x83900400 // precharge all; wait 7 cycles (set wait to tRP)

dcr_write conf_adr 0x0055 // INITPLR5

dcr_write conf_dta 0x85880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)

dcr_write conf_adr 0x0056 // INITPLR6

dcr_write conf_dta 0x85880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)

dcr_write conf_adr 0x0057 // INITPLR7

dcr_write conf_dta 0x85880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)

dcr_write conf_adr 0x0058 // INITPLR8

dcr_write conf_dta 0x85880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)

dcr_write conf_adr 0x0059 // INITPLR9

dcr_write conf_dta 0x81800032 // MRS - normal operation; wait 3 cycle (set wait to tMRD)(see **Note**)

dcr_write conf_adr 0x0021 // MCOPT2

dcr_write conf_dta 0x28000000 // Exit self refresh, Initiate preloaded sequence, Enable Controller

At this time, the DCEN bit of the MCIF0_MCOPT2 register must be set high to start auto refresh and to allow normal memory operations.

A typical initialization procedure for DDR2 operating at 133MHz, CAS Latency = 4, Burst Length = 4, Burst Type = sequential, Twr = 4, is as follows:

dcr_write conf_adr 0x00000050 // INITPLR0

dcr_write conf_dta 0x81B80000 // no-op; wait min 3 cycles

dcr_write conf_adr 0x00000051 // INITPLR1
dcr_write conf_dta 0x83900400 // precharge all; wait min 7 cycles (set wait to tRP)
dcr_write conf_adr 0x00000052 // INITPLR2
dcr_write conf_dta 0x81820000 // EMR2
dcr_write conf_adr 0x00000053 // INITPLR3
dcr_write conf_dta 0x81830000 // EMR3
dcr_write conf_adr 0x00000054 // INITPLR4
dcr_write conf_dta 0x 81810004 // EMR1; enable DLL, enable 75 Ohm ODT, enable DQS differential signaling.
dcr_write conf_adr 0x00000055 // INITPLR5
dcr_write conf_dta 0xBF800742 // MRS - reset DLL; wait 127 cycles (set wait to tMRD)(see **Note**)
dcr_write conf_adr 0x00000056 // INITPLR6
dcr_write conf_dta 0x83900400 // precharge all; wait 7 cycles (set wait to tRP)
dcr_write conf_adr 0x00000057 // INITPLR7
dcr_write conf_dta 0x8F880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)
dcr_write conf_adr 0x00000058 // INITPLR8
dcr_write conf_dta 0x8F880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)
dcr_write conf_adr 0x00000059 // INITPLR9
dcr_write conf_dta 0x8F880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)
dcr_write conf_adr 0x0000005A // INITPLR10
dcr_write conf_dta 0x8F880000 // Auto-refresh; wait 11 cycles (set wait to tRFC)
dcr_write conf_adr 0x0000005B // INITPLR11
dcr_write conf_dta 0x81800642 // MRS - normal operation; wait 3 cycle (set wait to tMRD) (see **Note**)
dcr_write conf_adr 0x0000005C // INITPLR12
dcr_write conf_dta 0x 81810384 // EMR1; OCD Calibration Default
dcr_write conf_adr 0x0000005D // INITPLR13
dcr_write conf_dta 0x 81810004 // EMR1; OCD Calibration Mode Exitdcr_write conf_adr 0x00000021 // MCOPT2
dcr_write conf_dta 0x28000000 // Exit self refresh, Initiate pre-loaded sequence, Enable Controller

Note: Device-specific values should be set here.

Figure 22-26 illustrates the field setting combinations of the MCIF0_INITPLRn register set.

Figure 22-26. Field Usage of MCIF0_INITPLRn Register Set for DDR2

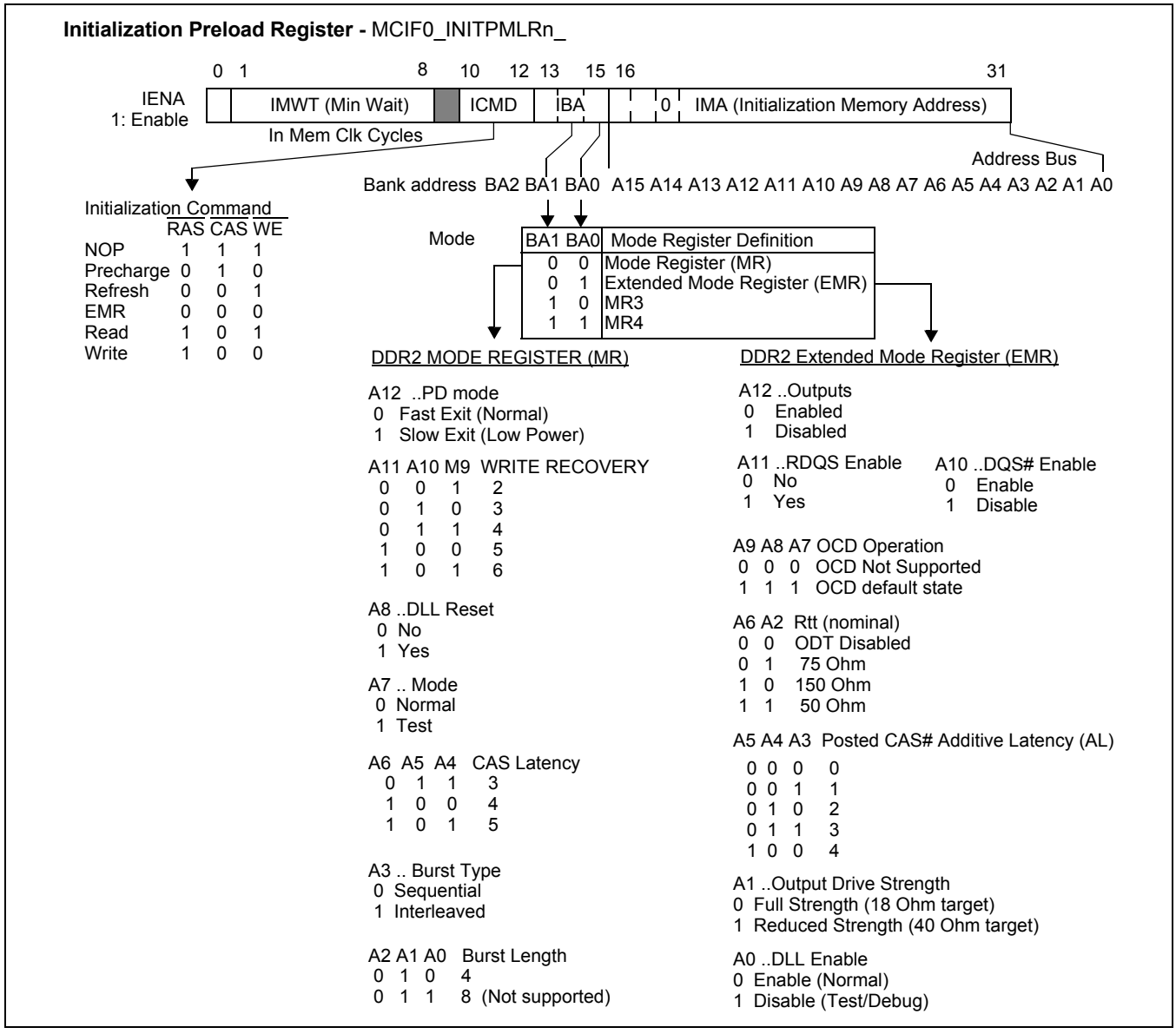


Table 22-16. MCIF0_INITPLRn Register Configuration for DDR1 Memory

	IENA (Enable)	IMWT (Minimum Wait)	Reserved	ICMD (Iniit. Command)				IBA	Reserved	IMA (Init. Memory Address)
				Command*	RAS	CAS	WE	BA2:0		MemAddr14:0
MCIF0_INITPLR0	1	400ns	0	NOP	1	1	1	0b000	0	0b0000000000000000
MCIF0_INITPLR1	1	tRP/tCK**	0	Precharge All	0	1	0	0b000	0	0b0000010000000000
MCIF0_INITPLR2	1	tMRD/tCK	0	Write EMR, Enable DLL	0	0	0	0b001	0	0b0000000000000000
MCIF0_INITPLR3	1	255 cycles, 0b11111111	0	Write MR, Reset DLL	0	0	0	0b000	0	See *Table 22-17*.
MCIF0_INITPLR4	1	tRP/tCK	0	Precharge All	0	1	0	0b000	0	0b0000010000000000
MCIF0_INITPLR5	1	tRFC/tCK	0	Auto Refresh	0	0	1	0b000	0	0b0000000000000000
MCIF0_INITPLR6	1	tRFC/tCK	0	Auto Refresh	0	0	1	0b000	0	0b0000000000000000
MCIF0_INITPLR7	1	tRFC/tCK	0	Auto Refresh	0	0	1	0b000	0	0b0000000000000000
MCIF0_INITPLR8	1	tRFC/tCK	0	Auto Refresh	0	0	1	0b000	0	0b0000000000000000
MCIF0_INITPLR9	1	tMRD/tCK	0	Write MR, Clear Reset DLL	0	0	0	0b000	0	See *Table 22-17*.
MCIF0_INITPLR10	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR11	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR12	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR13	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR14	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR15	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000

*EMR - DDR Memory Extended Mode Register
MR - DDR Memory Mode Register
**tCK= MemClkOut period

Table 22-17. Mode Register Configuration for DDR1 Memory

Address Bits	A14:9	A8 (DLL Reset)	A7 (Mode)	A6:4 (CAS Latency)	A3 (Sequential)	A2:0 (Burst Length)
MCIF0_INITPLR3[IMA]	0b000000	1	0	CL*	0	0b010
MCIF0_INITPLR9[IMA]	0b000000	0	0	CL*	0	0b010

*CL (CAS Latency in cycles)
0b010 = 2
0b011 = 3
0b110 = 2.5

Table 22-18. MCIF0_INITPLLn Register Configuration for DDR2 Memory

	IENA (Enable)	IMWT (Minimum Wait)	Reserved	ICMD (Iniit. Command)				IBA	Reserved	IMA (Init. Memory Address)
				Command*	RAS	CAS	WE	BA2:0		MemAddr14:0
MCIF0_INITPLR0	1	400ns	0	NOP	1	1	1	0b000	0	0b0000000000000000
MCIF0_INITPLR1	1	tRP/tCK***	0	Precharge All	0	1	0	0b000	0	0b0000010000000000
MCIF0_INITPLR2	1	tMRD/tCK	0	Write EMR2	0	0	0	0b010	0	0b0000000000000000
MCIF0_INITPLR3	1	tMRD/tCK	0	Write EMR3	0	0	0	0b011	0	0b0000000000000000
MCIF0_INITPLR4	1	tMRD/tCK	0	Write EMR1, Enable DLL	0	0	0	0b001	0	0b0000000000000100**
MCIF0_INITPLR5	1	255 cycles 0b11111111	0	Write MR, Reset DLL	0	0	0	0b000	0	See *Table 22-19*.
MCIF0_INITPLR6	1	tRP/tCK	0	Precharge All	0	1	0	0b000	0	0b0000010000000000
MCIF0_INITPLR7	1	tRFC/tCK	0	Auto Refresh	0	0	1	0b000	0	0b0000000000000000
MCIF0_INITPLR8	1	tRFC/tCK	0	Auto Refresh	0	0	1	0b000	0	0b0000000000000000
MCIF0_INITPLR9	1	tRFC/tCK	0	Auto Refresh	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR10	1	tRFC/tCK	0	Auto Refresh	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR11	1	tMRD/tCK	0	Write MR, Clear Reset DLL	0	0	0	0b000	0	See *Table 22-19*.
MCIF0_INITPLR12	1	tMRD/tCK	0	Write EMR 1, OCD calibration default	0	0	0	0b001	0	See *Table 22-19*.
MCIF0_INITPLR13	1	tMRD/tCK	0	Write EMR 1, OCD calibration exit	0	0	0	0b001	0	See *Table 22-19*.
MCIF0_INITPLR14	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000
MCIF0_INITPLR15	0	0b0000000	0	-	0	0	0	0b000	0	0b0000000000000000

*EMR1—DDR Memory Extended Mode Register 1
EMR2—DDR Memory Extended Mode Register 2
EMR3—DDR Memory Extended Mode Register 3
MR—DDR Memory Mode Register
**EMR1 = 0b0000000000000100 enables 75Ω ODT. Depending on the board layout and number of ranks, 150Ω may be more appropriate.
***tCK= MemClkOut period

Table 22-19. Mode Register Configuration for DDR2

Address Bits	A14:13	A12	A11:9 (Write Recovery)	A8 (DLL Reset)	A7 (Mode)	A6:4 (CAS Latency)	A3 (Sequential)	A2:0 (Burst Length)
MCIF0_INITPLR5[IMA]	00	0	tWR/tCK**	1	0	CL*	0	010
MCIF0_INITPLR11[IMA]	00	0	tWR/tCK	0	0	CL*	0	010

*CL (CAS Latency in cycles)
0b010 = 2
0b011 = 3
0b100 = 4
0b101 = 5
0b110 = 6
0b111 = 7
**tCK= MemClkOut period

22.2.8.14 Read DQS Delay Control Register (MCIF0_RQDC)

During the SDRAM Read phase, MCIF0_RQDC controls the delay between the incoming DQS signal and the clocking of the first Read flip-flop stage. It permits compensating for driver and external circuit delays and enables meeting various memory device requirements. The DQS is delayed from 0 to 255/128th of the DDR 1x half clock cycle and is used to center and capture data at either the rising edge or falling edge of the DQS during read sample stage 1. An 80° (0x38) phase shift is typical with Read DQS in the center of the data window. Read tuning must be performed at initialization time as described in *SDRAM Initialization* on page 793 in order select a specific DQS delay. The following figure describes the MCIF0_RQDC register bits.

Note: This register must be initialized only after the Master delay line has completed an initial calibration cycle as indicated by MCIF0_DLCR[DLCS] (bit 6 only) = 1, and after the memory initialization sequence has completed as indicated by MCIF0_MCSTAT[MIC] (bit 0) = 1.

Figure 22-27. Read DQS Delay Control Register (MCIF0_RQDC)

0	RQDE	Internal DQS Delay Mechanism Enable: 0 Disabled 1 Enabled	If enabled, the internal DQS delay mechanism (as configured by RQFD) is used to phase shift the incoming read DQS signals for read data capture. If disabled, an external mechanism must be used to phase shift the incoming read DQS signals for read data capture.
1:23		Reserved	
24:31	RQFD	Fractional (Fine) Delay: Valid range is from 0/128ths (h'000) to 255/128ths (h'0FF).	Defines the delay as a fraction in 1/128ths of a DDR 1x half clock cycle MCIF0_RQDC, when enabled, must be programmed with the appropriate delay to enable reliable capture of the read data into Stage 1 of the read sample logic. **Note:** This value is typically programmed for an 80 degree phase shift (56/128*180° = 78.75° where 56 = 0x38) and is based on the valid data window of the memory device, such that the Read DQS is positioned in the center of the valid data window for a single bit of data.

22.2.8.15 Read Feedback Delay Control Register (MCIF0_RFDC)

MCIF0_RFDC controls the feedback sampling process. The following figure describes the MCIF0_RFDC register bits.

Note: This register must be initialized only after the Master delay line has completed an initial calibration cycle as indicated by MCIF0_DLCR[DLCS] (bit 6 only) = 1. Similarly, before this register is written, the memory must be initialized and functional, as indicated by MCIF0_MCSTAT[MIC] (bit 0) = 1. If the memory has not been initialized or is in self-refresh mode, writing this register can stall the DCR logic and prevent further operations.

Figure 22-28. Read Feedback Delay Control Register (MCIF0_RFDC)			
0	ARSE	Auto-detect Read Sample Cycle Enable on Register update: 0 Enabled 1 Disabled	If enabled, a write to this register triggers a *dummy* read calibration access to each of the enabled ranks to determine the appropriate setting of MCIF0_RDCC[RDSS] based on the sampled and oversampled arrival of the feedback. Note: MCIF0_RDCC[RSAE] must also be enabled so the result of this calibration read sequence can update MCIF0_RDCC[RDSS]; otherwise the calibration reads will occur but MCIF0_RDCC[RDSS] will not be updated.
1:8		Reserved	
9:15	RFOS	Guard Band Feedback Fractional Oversample: Valid range is from 0/128ths (0x00) to 127/128ths (0x7F) DDR 1x cycles. 0x00 0.50 0x0B 0.45 0x18 0.40 0x25 0.35 0x32 0.30 0x3F 0.25 0x4B 0.20 0x58 0.15 0x65 0.10 0x72 0.05	Defines the oversample delay as a fraction, in the range of 0/128ths to 127/128ths of a DDR 1x half clock cycle. This register is programmed to select the desired oversample amount. The oversample window is defined by the remaining DDR 1x half clock cycle after the fractional oversample delay is set. RFOS = DEC2HEX(127 − Oversample*256) where 0 ≤ Oversample < 128/256 DDR 1x clock cycles.
16:20		Reserved	
21:31	RFFD	Feedback (FDBK) Fractional Delay in DDR1x half clock cycles. Valid range is from 0/128ths (0x000) to 1023/128ths (0x3FF).	Defines the Feedback delay as a fraction in the range of 0/128ths to 1279/128ths of a DDR 1X half-clock cycle (0x4FF). The delay is added to the MemDCFdbkR input feedback signal. This delay is accomplished in two stages: A coarse stage using bits 21:24, and a fine stage using bits 25:31. The coarse delay is done prior to the feedback signal leaving the chip. The fine delay happens after the feedback signal is received. This split delay must be taken into account when observing the timing of the feedback signal outside the chip (with respect to the receive DQS signals).

This register is programmed in conjunction with MCIF0_RQDC (which controls the delay of the DQS input signal) to enable read data capture.

To reduce read latency to its lowest possible value, the following steps must be performed:

1. Set and maintain the required oversample setting MCIF0_RFDC[RFOS] (bits 9:15) for the application.

   Note: This value must be set once and must remain unchanged for all subsequent accesses to the MCIF0_RFDC register.

2. Increment MCIF0_RQDC[RQFD] across the delay range while reading test patterns to determine the valid pass/fail window for DQS placement.
3. Repeat Step 2, incrementing MCIF0_RFDC[RFFD] across the delay range.
4. Adjust the position of MCIF0_RQDC[RQFD] and MCIF0_RFDC[RFFD] (in that order) so they are centered within the largest passing window found during the Step 2–Step 3 loop.

See *Read Data Capture Control Register (MCIF0_RDCC)* on page 784 and *SDRAM Initialization* on page 793 for additional information.

22.2.8.16 Read Data Capture Control Register (MCIF0_RDCC)

MCIF0_RDCC enables compensating for driver and external circuit delays and meeting various memory device requirements. The value in MCIF0_RDCC[RDSS] defines the cycle in which data is sampled during read sample Stage 2. T1 is the earliest sampling cycle; T4 is the latest sampling cycle.

If Auto-Update mode is enabled, the latency is reduced to the minimum because of the feedback signal arrival and the oversampling circuit. These define a "guard band" in which timing drift caused by voltage or temperature variations can be detected, resulting in an update to the MCIF0_RDCC[RDSS] value. The following figure describes the MCIF0_RDCC register bits.

Figure 22-29. Read Data Capture Control Register (MCIF0_RDCC)

0:1	RDSS	Read Sample Cycle Select: 00 T1 Sample 01 T2 Sample 10 T3 Sample 11 T4 Sample	
2:30		Reserved	
31	RSAE	Read Sample Cycle Auto-Update Disable: 0 Enabled 1 Disabled	

The RDSS bit sets the cycle (with respect to the read CAS latency of the device) in which the data/feedback are valid for sampling. The bit value is determined by the *round-trip* delay within the memory subsystem from clock to DQS on a read operation.

Data captured by DQS in Stage 1 is held for four MemClkOut cycles. Refer to *Figure 22-17* for a waveform diagram showing Stage 1. The sample cycle T1, T2, T3 or T4 specify when the read data is re-sampled by the internal controller clock (DDR 1x clock) in Stage 2. Even though there are four possible sample cycles, only two are available for sampling. The currently configured sample cycle is labeled the BASE cycle, and the alternate later sample cycle is labeled the PLUS1 cycle. For best read latency performance it is desirable to select the earliest sample cycle.

When read sample cycle auto-update is enabled (MCIF0_RDCC[RSAE] = 0), the BASE cycle is automatically updated every refresh interval or by a write to the MCIF0_RQDC[RFDC] register. At initialization time, the initial sample cycle is set to T1 and the DDR SDRAM controller automatically updates the sample cycle based on an oversample of the data and feedback pulse. The location of the sample time with respect to the DQS arrival is checked with the first read data in a four cycle burst. If changes in temperature and/or voltage conditions should delay the DQS arrival beyond a set period of time (MCIF0_RFDC[RFOS]), then the DDR SDRAM controller, during the next memory refresh period, selects the next highest (Tn+1) MCIF0_RDCC[RDSS] setting.

The feedback signal and Stage 1 read data are sampled in the on-time and oversample domains. If the feedback signal is captured ("hits") in both the on-time and oversample domains, data is sampled in the BASE cycle as defined by RDCC[RDSS]. However, if the sampled feedback signal "hits" in the on-time domain, but "misses" in the oversample domain, read data is not sampled until the next DDR 1x clock cycle and RDCC[RDSS] is incremented at the next refresh cycle. These two possible sample cycles define the capture window in which read data may be captured in the DDR 1x clock domain. MCIF0_RDCC[RDSS] does revert back to it's initial BASE setting provided that 100% of the feedback samples between refresh cycles "hit" in both the on-time and oversample domains. This process continues providing an automatic means of adjusting and minimizing read data capture latency.

When read sample cycle auto-update is disabled (MCIF0_RDCC[RSAE] = 1), the BASE cycle is selected by initialization software and may be updated periodically by software. Updates to MCIF0_RDCC[RDSS] should be made based on the status reported in the Runtime Tracking Status Register (MCIF0_RTSR).

22.2.8.17 Delay Line Calibration Register (MCIF0_DLCR)

MCIF0_DLCR provides the software with a hardware-based method of calibrating the various programmable delay lines (DLLs) available for use in the DDR SDRAM memory controller. The logic associated with this register starts to operate when reset is removed; as a result, the reset value is not a fixed value. The following figure describes the MCIF0_DLCR register bits.

Figure 22-30. Delay Line Calibration Register (MCIF0_DLCR)

0	DLCM	Delay Line Calibration Mode: 0 Automatic 1 Manual	This bit specifies the delay line calibration mode.
1:3		Reserved	
4	DLCR	Delay Line Calibration Request: 0 Idle 1 Calibrate Delay Line	Writing a 1 to this bit initiates the programmable delay line calibration procedure, if configured for Manual calibration mode. If switching from Manual to Automatic mode, writing a 1 to this bit starts the Automatic calibration process if that process is not running at the time. This bit does not remain set. Instead, it *pulses* for 1 clock cycle. The delay line calibration process is automatically initiated by the controller following power-on reset, independent of the Calibration mode.
5:7	DLCS	Delay Line Calibration Status: 000 Delay Line Calibration not run 001 Delay Line Calibration in progress 010 Delay Line Calibration complete (manual mode) 011 Delay Line Calibration done and continues (auto mode) 100 Delay Line Calibration error	This bit specifies the delay line calibration status.
8:22		Reserved	
23:31	DLCV	Delay Line Calibration Value	This 9-bit binary encoded value indicates the number of delay steps in a single DDR 1x half clock cycle. Bit 23 is the msb; bit 31 is the lsb.

The calibration process is automatically initiated by the controller after power-on reset. The resulting status and calibration values in the register are updated after calibration is complete. If reading this register following power-on reset, the status of the auto-initiated sequence is returned.

This feature provides a voltage- and temperature-compensated method of maintaining the fractional delay settings for the write channel (WRDTR), read DQS delay controls (RQDC), and read feedback delay controls (RFDC).

The implementation is a master/slave delay line arrangement in which the master delay line is continuously calibrated (if configured for Automatic mode) to determine the number of delay steps in a single DDR 1x half clock cycle. The number of delay steps are then used in conjunction with the programmed fractional delay settings to position the respective interface/signals/clock at the programmed delay step. The slave delay lines (for each of the fractional settings) are then updated during every auto-refresh interval to dynamically adjust the programmable delay to compensate for voltage and temperature shifts.

22.2.8.18 Memory Clock Timing Register (MCIF0_CLKTR)

MCIF0_CLKTR enables compensating for driver and external circuit delays and meeting various memory device requirements. The following figure describes the MCIF0_CLKTR register bits.

Figure 22-31. Memory Clock Timing Register (MCIF0_CLKTR)

0:1	CKTR	Memory Clock Phase Coarse timing adjustment: 00 Advance 0 degrees 01 Advance 90 degrees 10 Advance 180 degrees 11 Advance 270 degrees	This bit enables advancing or delaying the clocks supplied to the memory with respect to the controller 1x clock.
2:31		Reserved	

22.2.8.19 Write Data/DM/DQS Timing Register (MCIF0_WRDTR)

MCIF0_WRDTR enables compensating for driver and external circuit delays and meeting various memory device requirements. The following figure describes the MCIF0_WRDTR register bits.

Figure 22-32. Write Data/DM/DQS Timing Register (MCIF0_WRDTR)

0	WDPT	Write Preamble length selection 0 Reserved 1 1 cycle	This bit specifies the write preamble length.
1:2		Reserved	
3	LLWP	Low Latency Write Path Selection 0 One cycle added to support all WDTR settings 1 No cycles added, disallowing WDTR settings 000 and 001	
4:6	WDTR	Write Data/DQS/DM Phase Coarse timing adjustment. 000 270 degree advance 001 180 degree advance 010 90 degree advance 011 0 degree advance (nominal DQSS timing) 100 90 degree delay 101 180 degree delay 110 270 degree delay 111 reserved	This bit permits advancing or delaying these signals with respect to the controller DDR 1x clock. tDQSS: Write command to first DQS latching transition
7:19		Reserved	
20	WDAM	Write Data/Strobe/Mask Fine timing adjustment mode: 0 Reserved 1 Fractional DDR 1x half cycle (specified in 1/128ths)	This bit specifies the write data/strobe/mask fine timing.
21:23		Reserved	
24:31	WDFD	Fractional (Fine) Delay: Valid range is from 0x00 0/128ths to 0x3F 63/128ths.	Defines the delay as a fraction, in 1/128ths, of a DDR 1x half clock cycle. Note: The total fine delay must not exceed 1/4 of a DDR 1x clock cycle. Values that exceed this limit are not supported and may result in controller malfunction.

22.2.8.20 SDRAM Timing Register 1 (MCIF0_SDTR1)

The values in MCIF0_SDTR1 control the control and data bus timings, to allow for varying system topologies and speed requirements. When set to 1, LDOF (leadoff) forces every memory command to be two cycles in length, with a single chip select (CS) cycle. The following figure describes the MCIF0_SDTR1 register bits.

Figure 22-33. SDRAM Timing Register 1 (MCIF0_SDTR1)

0	LDOF	Command Leadoff Enable - 0 0 - Single Cycle Commands 1 1 - Two Cycle Commands with one cycle CS	This bit specifies how many cycles the command is driven. When set to two cycles, the command is driven one cycle prior to BankSeln (chip select).
1:7		Reserved	
8:11	RTW	Read to Write Bus Turnaround Delay 0010 2 cycles 0011 3 cycles Others - Reserved	This bit specifies the read to write bus turnaround delay.
12:15	WTWO	Write to Write Bus Turnaround Delay (other CS) 0000 0 cycles (normal operation) 0001 1 cycle (one cycle of data bus delay) Others reserved.	This bit specifies the write to write bus turnaround delay
16:19	RTRO	Read to Read Bus Data Turnaround Delay (other CS) 0001 1 cycle (normal operation, 1 bus transfer cycle) 0010 2 cycles (2 bus transfer cycles) Others reserved.	This bit specifies the read to read bus data turnaround delay.
20:31		Reserved	

22.2.8.21 SDRAM Timing Register 2 (MCIF0_SDTR2)

MCIF0_SDTR2 must be programmed to configure the DDR SDRAM device-specific timing parameters (expressed in units of clock cycles) required by the selected device. The industry standard designation of the specification value is noted in the table (tCL = CAS Latency, tAL = Additive Latency, and so on). The following figure describes the MCIF0_SDTR2 register bits.

Figure 22-34. SDRAM Timing Register 2 (MCIF0_SDTR2)

0:3	RCD	Activate to Read/Write Delay (tRCD, minimum) (cycles) 0001 1 0010 2 0011 3 0100 4 0101 5 Others Reserved.	
4:7	WTR	Write (data) to Read Command Delay (cycles) (for DDR2 = tWTR, for DDR1 = tCDLR) (minimum) 0001 1 0010 2 0011 3 0100 4 Others Reserved	
8:15	XSNR	Exit Self Refresh to Non-Read Command Delay (cycles) 00001000 8 00010000 16 00100000 32 01000000 64 Others Reserved	
16:19	WPC	Write Data to Precharge Delay (minimum) (cycles) 0010 2 0011 3 0100 4 0101 5 0110 6 Others Reserved, Recommended value is tWR + 1	
20:23	RPC	Read Data to Precharge Delay (minimum) (cycles) 0010 2 0011 3 0100 4 Others Reserved Recommended value is: 2 cycles = (Burst Length/2)	
24:27	RP	Precharge Command Period (tRP, minimum) (cycles) 0011 3 0100 4 0101 5 0110 6 0111 7 Others Reserved.	

28:31	RRD	Activate to Activate Delay (tRRD, same CS, minimum) (cycles) 0010 2 0011 3 Others Reserved.	

Notes:
1. WPC: This timing parameter is used, along with other timing values, to determine the number of memory cycles of delay between the issuing of a Write command and a following Precharge or Activate command to the same bank. (Other timing parameters, such as tRC, may further increase these limits.) The delays are calculated as follows:
(For DDR2 devices): Write to Precharge Delay = tCLW + tAL + tWPC + 2.
(For DDR1 devices): Write to Precharge Delay = tWPC + 3.
(For DDR2 devices): Write with AutoPC to Activate = tCLW + tAL + tWPC + tRP + 3.
(For DDR1 devices): Write with AutoPC to Activate = tWPC + tRP + 3.

2. RPC: This timing parameter is used, along with other timing values, to determine the number of memory cycles of delay between the issuing of a Read command and a following Precharge or Activate command to the same bank. (Other timing parameters, such as tRC, may further increase these limits.) The delays are calculated as follows:
(For all devices): Read to Precharge Delay = tAL + tRPC.
(For all devices): Read with AutoPC to Activate = tAL + tRP + 2.

22.2.8.22 SDRAM Timing Register 3 (MCIF0_SDTR3)

MCIF0_SDTR3 must be programmed to configure the DDR SDRAM device-specific timing parameters (expressed in units of clock cycles) required by the selected device. Settings outside these specifications may cause invalid operation. The following figure describes the MCIF0_SDTR3 register bits.

Figure 22-35. SDRAM Timing Register 3 (MCIF0_SDTR3)

0:2		Reserved	
3:7	RAS	Activate to Precharge Time (tRAS)	
8:10		Reserved	
11:15	RC	Activate to Activate/AutoRefresh Time (tRC, minimum)	
16:18		Reserved	
19:23	XSC	Exit Self Refresh and DLL Lock Delay (tXSC)	Count is in multiples of 16 clocks (recommended value is 01101). Some devices specify this value as XSRD. The suggested value corresponds to 208 cycles of delay. This delay is applied after CKE is set and before any INITR triggered commands may be applied to the memory.
24:25		Reserved	
26:31	RFC	Auto Refresh to Command Delay (tRFC)	Value is in clock cycles (0x10 default). This may vary depending on the SDRAM manufacturer and size. Example: For MemClkOut = 200MHz **256Mb 512Mb 1Gb 2Gb** **tRFC** 75ns 105ns 127.5ns 195ns **Cycles** 15 21 36 39 **RFC** 0x0F 0x15 0x1A 0x27

22.2.8.23 SDRAM Mode Register (MCIF0_MMODE)

MCIF0_MMODE must be programmed to a value that reflects the *operational* mode of the selected memory device. Some memories may define these values differently. See the selected memory specification to select the final value.

These values are not programmed into the memories. The programming of the memories is done by the MCIF0_INITPLRn register settings.

Typical values for DDR1 and DDR2 devices are shown.

Figure 22-36. SDRAM Mode Register (MCIF0_MMODE)			
0:19		Reserved, always 0s	
20:22	WR	Write Recovery Time (cycles) DDR1: Always set to 000 DDR2: 010 3 011 4 100 5 101 6 All other values Reserved.	
23:24		Reserved, always 0s	
25:27	DCL	Device CAS Latency (cycles) DDR1: 010 2 110 2.5 011 3 DDR2: 010 2 011 3 100 4 101 5 110 6 111 7 All other values Reserved	See the selected memory specification for the correct value for this field. The entries shown in this table are typical.
28	BTYP	Burst Type	Hardwired to 0, for sequential.
29:31	BLEN	Burst Length	Hardwired to 010, for burst length = 4. Burst length of 8 is not supported.

22.2.8.24 SDRAM Extended Mode Register (MCIF0_MEMODE)

MCIF0_MEMODE must be programmed to a value that reflects the *operational* mode of the selected memory device. Some memories may define these values differently. See the selected memory specification to select the final value.

It is recommended that Additive Latency be set to 0 cycles if Page Management is set to keep pages open after use. This reduces memory read latency in this mode.

These values are not programmed into the memories. The programming of the memories is done by the MCIF0_INITPLRn register settings.

Figure 22-37. SDRAM Extended Mode Register (MCIF0_MEMODE)			
0:11		Reserved	
12:18	EMU	Extended mode upper bits (optional)	
19	QOFF	Q-off 0 Output buffer enabled 1 Output buffer disabled	

20	RDQS	Read DQS Control (DDR2 only) 0 Disable 1 Reserved	
21	DQS#	Enable DQS differential signal (DDR2 only) 0 Enable 1 Disable	
22:24		Reserved	
25	RTT0	Memory Device Signal Termination Control Bits 25 and 29: 00 ODT Disabled 01 75 ohm 10 150 ohm 11 Reserved	
26:28	AL	Additive Latency in Cycles (DDR2 devices only) 000 0 (set this value for DDR1 devices) 001 1 cycle 010 2 cycles 011 3 cycles 100 4 cycles	
29	RTT1	Memory Device Signal Termination Control	LSB associated with bit 25 of this register.
30	DIC	Memory Device Driver Impedance Control 0 Normal 1 Weak (60%)	
31	DLLE	DLL Enable (0).	

22.2.8.25 ECC Error Status Register (MCIF0_ECCES)

MCIF0_ECCES, accessed at offset 0x98, tracks ECC-related errors encountered during SDRAM accesses to memory. Clearing individual bits of this error status register is possible on write. The following figure describes the MCIF0_ECCES register bits.

Figure 22-38. ECC Error Status Register (MCIF0_ECCES)

0:15	BnCE	Byte Lane n Corrected Error (where n = 0–15): 0 No Error 1 Error Occurred in Byte Lane n	Bits 0:7 indicate the byte lane of the first 64 data bits on the rising edge of the memory clock. Bytes 8:15 correspond to byte lanes 0:7 of the second beat of 64 data bits on the falling edge of the memory clock. A single-bit error can be reported on both the first and second beats of 64 data bits.
16:17	CKBER	Error Detected in Checkbits: 64-bit Mode 00 No Error 01 Reserved 10 Error in Checkbits 11 Reserved 32-bit Mode 00 No Error 01 Error in Lower Checkbits 10 Error in Upper Checkbits 11 Error in Upper and Lower Checkbits	
18	CE	1 Correctable Error	
19	UE	1 Uncorrectable Error	

20:23	BKnER	Error in Rank 0 (bit 20), Rank 1 (bit 21), Rank 2 (bit 22), or Rank 3 (bit 23): 0 No Error 1 Error Occurred	
24:31		Reserved	

22.2.8.26 Feedback Calibration Status Register (MCIF0_FCSR)

This register captures status associated with the automatic feedback calibration mechanism and is intended for diagnostic purposes only. This register is updated each time a write occurs to RFDC and auto-update is enabled.

Figure 22-39. Feedback Calibration Status Register (MCIF0_FCSR)

0:3	R0CNT	Rank 0 Latency Count	
4:7	R1CNT	Rank 1 Latency Count	
8:11	R2CNT	Rank 2 Latency Count	
12:15	R3CNT	Rank 3 Latency Count	
16	R0MOS	Rank 0 Miss Oversample	
17	R1MOS	Rank 1 Miss Oversample	
18	R2MOS	Rank 2 Miss Oversample	
19	R3MOS	Rank 3 Miss Oversample	
20:23	RSELV	Rank Select Vector: 1xxx Rank 0 used 01xx Rank 1 used 001x Rank 2 used 0001 Rank 3 used 0000 Default	
24:25	CDSS	Calibration RDSS Setting. Reference MCIF0_RDCC[RDSS] for encoded value decode. This field reflects the initial value of MCIF0_RDSS at the end of the feedback calibration register (MCIF0_RFDC) access.	
26:27	CMOS	Calibration Missed Oversample: 00 Hit Oversample 01 Missed Oversample 1x Reserved This field reflects the value of RxMOS following the feedback calibration register (MCIF0_RFDC) access as determined by the Rank Select Vector. General indication that data is initially taken from the second cycle of the data capture.	
28:31			Reserved

22.2.8.27 Run Time Tracking Status Register (MCIF0_RTSR)

This register captures status associated with the automatic tracking mechanism and is intended for diagnostic purposes only. This register is updated continuously over a period of time. The various bits of this register can be "cleared" to the Reset Value by writing zeros to the corresponding field to be cleared

Note: On the clock cycle following reset deassertion, tracking of the associated state machines begins and continues on each state transition. As such, the value read following reset will reflect the state transition history of the state machines being tracked and will not reflect the reset value indicated.

.

Figure 22-40. Run Time Tracking Status Register (MCIF0_RTSR)

0:11	TRK1SM	Tracking State Machine 1 State History Two bit, encoded state machine Each time the SM changes state, the new state is shifted into bits 0:1 and bits 0:9 are shifted to bits 2:11 enabling the last 6 state transitions to be captured. State history is read right to left starting with 10:11 and moving 2 bits at a time back to 0:1. 00 atbase state 01 missed state 10 atpls1 state 11 Reset value	
12:23	TRK2SM	Tracking State Machine 2 State History Two bit, state machine. Each time the SM changes state, the new state is shifted into bits 0:1 and bits 0:9 are shifted to bits 2:11 enabling the last 6 state transitions to be captured. State history is read right to left starting with 10:11 and moving 2 bits at a time back to 0:1. 00 tridle state 01 mos state 10 hos state 11 Reset value	
24	RMOS	Runtime Missed Oversample	
25	RHOS	Runtime Hit Oversample	
26:31			Reserved

22.2.9 SDRAM Initialization

The DDR2 SDRAM controller must be initialized before use. After a power-on reset, the memory controller controller drives Clock Enable (MemClkEn) low (0) until commanded to exit "power-on" mode.

The following power-up sequence should be executed prior to enabling the Memory Controller.

1. Check "Power-On" mode
   Verify that the controller is in the "power-on" mode by performing a read to DCR register bit MCIF0_MCOPT2[SREN]. This bit indicates that the controller has entered self refresh mode.
2. Initialize the controller registers
   After the chip has come out of reset, software is responsible for configuring the DDR SDRAM controller. The following registers must be programmed to complete the configuration step.

- MQ0_BnBAS
- MQ0_BAUL
- MQ0_BAUH
- MQ0_CF1L
- MQ0_CF1H
- MQ0_CFBHL
- MCIF0_MCOPTn
- MCIF0_MODTn
- MCIF0_CODT

- MCIF0_RTR
- MCIF0_MBnCF
- MCIF0_SDTRn
- MCIF0_MMODE
- MCIF0_MEMODE

Note: On systems with ECC, enable ECC generation (MCIF0_MCOPT1[MCHK] = 0b10).

3. Prepare for initialization of memory devices
   Software must configure the MCIF0_INITPLRn registers as specified by the memory device specification.
4. Execute initialization sequence
   To use the memory, software must enable an exit out of "power-on" mode by writing 0 to MCIF0_MCOPT2[SREN]. At the same time, the initialization sequence is armed by writing 1 to MCIF0_MCOPT2[IPTR]. This write will set CKE = 1 and the initialization sequence is triggered after a delay of tXSC (MCIF0_SDTR3[XSC]) cycles.
5. Wait for initialization sequence completion
   Software can poll on MCIF0_MCSTAT[MIC] to determine when the initialization sequence is complete. This bit is set to 1 when the sequence completes.
6. Enable the controller
   Software then sets MCIF0_MCOPT2[DCEN] to 1 to enable the DDR SDRAM controller to accept commands from the command queue and to enable auto-refresh operation.
7. Master delay line calibration complete
   The various programmable delay lines are calibrated immediately following power-on reset after the system clocks have stabilized. It is important that the calibration sequence complete prior to proceeding to the next step. The status of the calibration sequence can be determined by polling MCIF0_DLCR. When bit 6 is 1, the delay line calibration sequence is complete.
8. Setup oversample setting
   Configure the feedback oversample MCIF0_RFDC[RFOS]. Typically RFOS is set to 1/4 cycle. Enable Read Sample Cycle Auto-Update, MCIF0_RDCC[RDAE] = 0, and select sample cycle T1, MCIF0_RDCC[RDSS] = 00.

To have software select and manage the sample cycle MCIF0_RDCC[RDSS], disable auto-update, MCIF0_RDCC[RSAE] = 1. In order to select the correct sample cycle, software should read the status reported in the Runtime Tracking Status Register (MCIF0_RTSR).

9. Determine valid DQS placement and valid feedback pulse placement
   Determining valid DQS placement should be performed if using the internal DQS delay mechanism. To delay DQS internally set MCIF0_RQDC[RQDE] = 1.

   This configuration step requires a single dummy write to memory followed by several successive dummy reads from memory. Between reads, the position of the feedback pulse is shifted by incrementing MCIF0_RFDC[RFFD] and the delay of DQS shifted by incrementing MCIF0_RQDC[RQFD]. The goal is to find the combination of settings that provide the largest window of valid reads. The following pseudo code describes two methods (A and B) for programming MCIF0_RFDC[RFFD] and MCIF0_RQDC[RQFD]. Method A is preferred.

Pseudo Code for Method A:

```
ARRAY [Entire DQS Range] DQS_Valid_Window;    //initialized to all zeros
ARRAY [Entire FDBK Range] FDBK_Valid_Window;    //initialized to all zeros
MEMWRITE(addr, expected_data);
for (i = 0; i < Entire DQS Range; i++) {
    for (j = 0; j < Entire Feedback Range; j++) {
```

```
            MEMREAD(addr, actual_data);
            if (actual_data == expected_data) {
            DQS_Valid_Window[i] = 1;
            FDBK_Valid_Window[i][j] = 1;
           }
        }
    }
```

Once this sequence completes, the array, DQS_Valid_Window, contains 1's indicating where valid data was returned. The center-point of this window should be set in MCIF0_RQDC[RQFD], indicating the valid placement of DQS. Then, based on that value, the valid window for Feedback can be found in the FDBK_Valid_Window array. The center-point of this window should be set in MCIF0_RFDC[RFFD], indicating the valid placement of Feedback.

Pseudo Code for Method B:

```
MEMWRITE(addr, expected_data);
Initialize the DQS delay to 80 degrees (MCIF0_RQDC[RQFD]=0x38).
 for (j = 0; j < Entire Feedback Range; j++) {
    MEMREAD(addr, actual_data);
     if (actual_data == expected_data) {
        FDBK_Valid_Window[j] = 1;
     }
}
Set MCIF0_RFDC[RFFD] to the middle of the FDBK_Valid_Window.
for (i = 0; i < Entire DQS Range; i++) {
   MEMREAD(addr, actual_data);
   if (actual_data == expected_data) {
      DQS_Valid_Window[i] = 1;
    }
}
Set MCIF0_RRQDC[RQFD] to the middle of the FDBK_Valid_Window.
```

10. Complete RDSS configuration
    After the MCIF0_RQDC[RQFD] and MCIF0_RFDC[RFFD] values are set, read MCIF0_RDCC[RDSS] and MCIF0_RTSR[TRK1SM]. If the position of MCIF0_RTSR[TRK1SM] is "Atpls1" (0b10), increment the value in MCIF0_RDCC[RDSS] to the next T Sample window. If the position is "Atbase" (0b00), leave the MCIF0_RDCC[RDSS] setting as is.
11. **Configuration complete for systems without ECC** - At this point, the memory controller and SDRAM are ready for use. On systems with ECC, perform steps 12 through 15.
12. Initialize the memory containing the ECC check bits by writing the entire installed memory with data. Often the address is used as the data pattern but any data pattern can be used. The data cache can be enabled for the writes to speed up the memory initialization process. The processor generates 32 byte accesses on cache flushes.
13. Wait until the transaction completes so that the DDR SDRAM is idle. To ensure the last write completed, read the last address written and execute a memory synchronization instruction (msync). The DDR SDRAM controller maintains read/write order for accesses to the same address so the write must complete before the read.

14. Enable ECC checking and generation (MCIF0_MCOPT1[MCHK]=0b01 or MCIF0_MCOPT1[MCHK]=0b11).
15. **Configuration complete for systems with ECC** - At this point, the memory controller and ECC SDRAM is ready for use.

22.2.10 DDR to Memory Address Decode

The DDR SDRAM memory can be located within any 4GB aligned region that can be accessed from two PLB address spaces:

- Low Latency (LL) PLB bus: from 0x0 0000 0000 to 0x0 FFFF FFFF
- High Bandwidth (HB) PLB bus: from 0x8 0000 0000 to 0x8 FFFF FFFF

The range of addresses to which the DDR SDRAM responds within the programmed 4-GB aligned region can be further limited to the specific range of addresses for which physical memory is installed.

Note: Moving the memory map may cause overlap with other memory mapped PLB slaves.

The specific range of addresses to which the PLB slave will respond is defined and configured by the combination of the High Bandwidth PLB Base Address Register—upper 32 bits (MQ0_BAUH), and the Rank Base Address and size Registers (MQ0_BnBAS).

The PLB slave interface responds to any PLB access that targets a defined base address and lies within the physical memory region defined for the enabled memory rank (if the access type is supported).

22.2.11 DDR Slave Interface Options

The LL and HB DDR SDRAM PLB slave interface has several configurable options to allow users to change the way in which the DDR SDRAM responds to read and write requests for various conditions such as buffer full, address queue full, and read/write priority. These options are explained in the following sections and configured by means of the following registers:

- Configuration 1 Register (HB) (MQ0_CF1H)
- Configuration 1 Register (LL) (MQ0_CF1L)

22.2.12 Command Queues and Out-of-Order Operation on Memory Interface

Due to the nature of SDRAM memory devices, a read or write request often result in several commands being issued to the memory device for the management of memory device banks or pages as well as actual read and write commands.

For example, if the desired address is contained in a memory row that is not active, and another row of that bank is in the active state, then that bank must be precharged, and then the new row activated, before the desired read or write can be performed. Further, device timing restrictions require that specific minimum times pass between the precharge, activate and read/write commands. In a conventional controller, these periods of waiting are not used.

This controller attempts to use these delay periods to perform memory operations that are needed by upcoming memory requests. To do this, the controller maintains an 'order queue' of up to four pending read and write requests. This allows the controller to look ahead in the request sequence and perform memory commands out of order in a manner which reduces the wasted wait cycles mentioned above. Requests are maintained in this queue until the final read or write associated with the request is done. If a request requires multiple reads or writes to complete, these reads or writes are never interrupted by other reads or writes. However, precharges and activates associated with other pending requests may be interleaved with the multi-cycle read or write operation. This does not delay the read or write operation, because the memory is always operated in the burst-of-4 mode, allowing read or write commands only every other cycle.

The degree of out-of-order operation allowed is programmable. Reads or writes can be individually required to be 'in order', with only the Activate and Precharge commands allowed out of order. Further, another mode forces all operations to be serialized strictly in order. Refer to the MCIF0_MCOPT1 register bits RWOO, WOOO and DCOO to set the desired mode of operation.

Reads or writes within the same internal memory bank are never re-ordered, to prevent the possibility of reading stale data. Re-ordering within an internal bank offers little gain, as only a single row can be active at any one time.

Refresh and other special commands bypass the queues.

22.2.13 Reordering Algorithm

During each memory clock cycle, one of the up to four pending read/write requests can cause a memory device command to be issued. In general, the command for the oldest request in the queue is given priority. The oldest valid command which can be performed in the next cycle is picked.

For example, if the oldest request is requiring an activate row command, and the memory timing does not allow that to happen at this time, then the next oldest request is evaluated. This is repeated until a valid request can result in a memory command, or all valid requests are exhausted (in which case an NOP is issued to the memory devices). This decision process is made in one memory clock cycle. Logic in the controller mimics the state of each memory device using the timing values programmed in the DCR registers (MCIF0_MCOPT1-2, MCIF0_SDTR1-3, MCIF0_MMODE, and MCIF0_MEMODE registers).

A simple example is that of two read requests in the request queue, and all memory banks in the idle state. In cycle one, the oldest read request results in an activate command to the desired bank and row. Assume a tRRD of two cycles, and a tRCD of four cycles. Because of these time delays, no command can be issued in the second cycle. In the third cycle, the activate command of the newer request can be issued to the memory device. Now both memory banks are awaiting the completion of tRCD delays, so no command may be issued in the forth cycle. During the fifth cycle, a read command may be issue to the first row, as tRCD has expired. Two cycles later (seventh cycle), the read associated with the newer request may be issued to the memory. If additional requests enter the queue that target non-conflicting banks, additional memory commands for the new requests can be interleaved with the above command sequence.

If write or read out-of-order operation is enabled, then a newer pending read or write may be done ahead of an older read or write that is waiting on a bank activate or precharge to complete.

22.2.14 Page Management

The DDR SDRAM controller supports page mode operation with bank interleaving (BA2:0) and maintains up to 32 open pages in the memory subsystem. Subsequent accesses that target an open page result in a page hit.

.The controller has three selectable page modes. The first mode (CLOSE) always closes pages after each read or write (using auto-precharge). The second mode (AUTO) closes pages after each read or write, unless the next command within the command queue will 'hit' the same page. The third mode (OPEN) keeps pages open until a read or write for another page (row) is queued. All pages are closed when the DDR SDRAM control logic issues a precharge all command to the DDR SDRAM in preparation for a refresh cycle.

The SDRAM page size for page hits varies, depending upon the programmed addressing mode, which is derived from the DDR SDRAM organization. *Table 22-20* and *Table 22-21* describe the relationship of the address mode (MCIF0_MBnCF[M_AM]) to the page size:

Table 22-20. 32-Bit SDRAM Page Size

Address Mode	Page Size
0 or 5	1KB
1 or 6	2KB
2 or 7	4KB
3 or 8	8KB
4 or 9	16KB

Table 22-21. 64-Bit SDRAM Page Size

Address Mode	Page Size
0 or 5	2KB
1 or 6	4KB
2 or 7	8KB
3 or 8	16KB
4	32KB

22.2.14.1 Physical Address-to-Memory Address Mapping

The DDR SDRAM controller supports various 4- and 8-bank DDR SDRAM device densities of 64Mb to 4Gb, in x8 and x16 organizations. The addressing configuration associated with the specific device(s) organization and density determines the corresponding addressing mode(s), independent of the packaging of those devices (DIMM, SODIMM, or Planar).

Table 22-22, and *Table 22-23* on page 799 summarize the supported addressing modes for DDR SDRAM densities and configurations required, for a 32-bit and a 64-bit data bus, respectively.

Row and Column addresses are generated based on the organization of the SDRAM banks as defined in the memory configuration registers (MCIF0_MB0CF and MB1CF). *Table 22-22*, and *Table 22-23* on page 799 show the mapping of the physical address to memory address for all memory accesses based on the configured mode for the associated rank; the DDR SDRAM device organization is row × column (internal banks).

N = 11, 12, 13, 14, or 15 (N = Number of Rows)

Table 22-22. 32-Bit DDR SDRAM Addressing Modes

Mode	SDRAM Config		BA2	BA1	BA0	MA14	MA13	MA12	MA11	MA10 /AP	MA9	MA8	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
0	Nx8 (4)	Row		24	25	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
		Col		24	25					AP			26	27	28	29	30	31	32	33
1	Nx9 (4)	Row		23	24	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
		Col		23	24					AP		25	26	27	28	29	30	31	32	33
2	Nx10 (4)	Row		22	23	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
		Col		22	23					AP	24	25	26	27	28	29	30	31	32	33
3	Nx11 (4)	Row		21	22	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		Col		21	22				23*	AP*	24	25	26	27	28	29	30	31	32	33
4	Nx12 (4)	Row		20	21	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		Col		20	21			22*	23*	AP*	24	25	26	27	28	29	30	31	32	33
5	Nx8 (8)	Row	23	24	25	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
		Col	23	24	25					AP*			26	27	28	29	30	31	32	33

Table 22-22. 32-Bit DDR SDRAM Addressing Modes

Mode	SDRAM Config		BA2	BA1	BA0	MA14	MA13	MA12	MA11	MA10 /AP	MA9	MA8	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
6	Nx9 (8)	Row	22	23	24	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
		Col	22	23	24					AP		25	26	27	28	29	30	31	32	33
7	Nx10 (8)	Row	21	22	23	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		Col	21	22	23					AP*	24	25	26	27	28	29	30	31	32	33
8	Nx11 (8)	Row	20	21	22	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		Col	20	21	22				23*	AP*	24	25	26	27	28	29	30	31	32	33
9	Nx12 (8)	Row	19	20	21	4	5	6	7	8	9	10	11	40	13	14	15	16	17	18
		Col	19	20	21			22*	23*	AP*	24	25	26	27	28	29	30	31	32	33

Table 22-23. 64-Bit DDR SDRAM Addressing Modes

Mode	SDRAM Config		BA2	BA1	BA0	MA14	MA13	MA12	MA11	MA10 /AP	MA9	MA8	MA7	MA6	MA5	MA4	MA3	MA2	MA1	MA0
0	Nx8 (4)	Row		23	24	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
		Col		23	24					AP			25	26	27	28	29	30	31	32
1	Nx9 (4)	Row		22	23	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
		Col		22	23					AP		24	25	26	27	28	29	30	31	32
2	Nx10 (4)	Row		21	22	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		Col		21	22					AP	23	24	25	26	27	28	29	30	31	32
3	Nx11 (4)	Row		20	21	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		Col		20	21				22*	AP*	23	24	25	26	27	28	29	30	31	32
4	Nx12 (4)	Row		19	20	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
		Col		19	20			21*	22*	AP*	23	24	25	26	27	28	29	30	31	32
5	Nx8 (8)	Row	22	23	24	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
		Col	22	23	24					AP			25	26	27	28	29	30	31	32
6	Nx9 (8)	Row	21	22	23	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		Col	21	22	23					AP		24	25	26	27	28	29	30	31	32
7	Nx10 (8)	Row	20	21	22	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		Col	20	21	22					AP	23	24	25	26	27	28	29	30	31	32
8	Nx11 (8)	Row	19	20	21	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
		Col	19	20	21				22*	AP*	23	24	25	26	27	28	29	30	31	32
9	Nx12 (8)	Row	18	19	20	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
		Col	18	19	20			21*	22*	AP*	23	24	25	26	27	28	29	30	31	32

Notes:

*****Column address bit 10 sent out on MA11 for 13 x 11 (4) parts.

**Blank entries are "don't care "for an addressing mode.

*******AP is driven based on the specific type of read or write access occurring. With page mode enabled, AP = 0 for all read and write commands. With page mode disabled, AP = 0 for reads/writes that are part of a sequential burst access and AP = 1 for the last read/write of any access.

22.2.15 SDRAM Commands and Operations

The DDR SDRAM controller supports the following commands:

- Read
- Write
- Activate

- Single-Bank Precharge
- Read/Write with Auto-Precharge
- Precharge All Banks
- Auto CAS Before RAS (CBR) Refresh
- Self-Refresh (Hardware and Software Initiated)

22.2.15.1 DDR SDRAM Timing Parameters

Programmable memory timing support provides for flexibility at the system level. As a result, memory timing parameters can be adjusted to support faster SDRAM technologies as they become available, maximizing the performance of the SDRAM memory subsystem.

Note: All SDRAM timing diagrams show only cycle-based programmable timing parameters. Do not infer AC-specific timing information based on the timing diagrams.

Table 22-24. SDRAM Memory Timing Parameters

Name	Function	Description
LDOF	Leadoff Cycle Enable	Make all memory commands two cycles, to allow additional command setup time before a Chip Select (Rank Select).
RCD	Activate to Read/Write Delay	Minimum number of clock cycles from an Activate Command to a Read Command.
WTR	Write (data) to Read Delay	Memory Internal Write to Read Delay in clock cycles. Called WTR or CDLR in memory specifications.
RTW	Read to Write Data Bus Turnaround Time	Delay for the memory data bus to switch directions between a read a a write operation.
WTWO	Write to Write Data Bus Turnaround Time	Data Bus delay between a write to one memory device to a write to another memory device. This is usually set to zero.
RTRO	Read to Read Data Bus Turnaround Time	Data Bus delay between a read from one memory device to a read from another memory device. This is usually set to one cycle.
WPC	Write Data Out to Precharge Delay	Delay from the last data out of a write operation to a Precharge Command to the same internal bank. This is usually set to the write recovery time (tWR) + 1.
RPC	Read Data to Precharge Delay	Delay from the last data of a read operation to a Precharge Command to the same internal bank. This is usually set to one-half the burst length or 2.
RP	Precharge to Activate Delay	Sets the minimum number of clock cycles from a Precharge Command to an Activate Command.
RRD	Bank A Activate to Bank B Activate	Module Bank-to-Bank Activate Delay.
RAS	Activate to Precharge Time	The minimum time delay between the Activation of a row to the Precharge of the same row.
RC	Activate to Activate Delay	The minimum row cycle time, or delay between Activate commands to the same internal bank.
XSC	Self Refresh Exit Delay	Minimum number of clock cycles until a command to memory is allowed following initial power on and self refresh exit (ClkEn = 1). Usually set to 200+ clocks to allow for DLL to lock.
RFC	Auto Refresh Cycle Time	The minimum time from an auto refresh command to any other memory command.
WR	Write Recovery Time (Mode Reg)	The minimum write recovery time, to be programmed into the Memory Mode Register at initialization. This field is set to zero for some memory types.

Table 22-24. SDRAM Memory Timing Parameters (Continued)

DCL	Cas Latency (Mode Reg)	The Cas Latency to be programmed into the Memory Mode Register at initialization. Refer to the memory device specification.
AL	Additive Latency	For DDR2 devices only. Additive latency permits the issuance of a CAS (read or write) command early. The RAS to CAS delay becomes tRCD - tAL.

The following timing diagrams illustrate the relationship of the programmable timings for activate, read, write, and precharge commands:

Figure 22-41. Activate - Read - Precharge - Activate

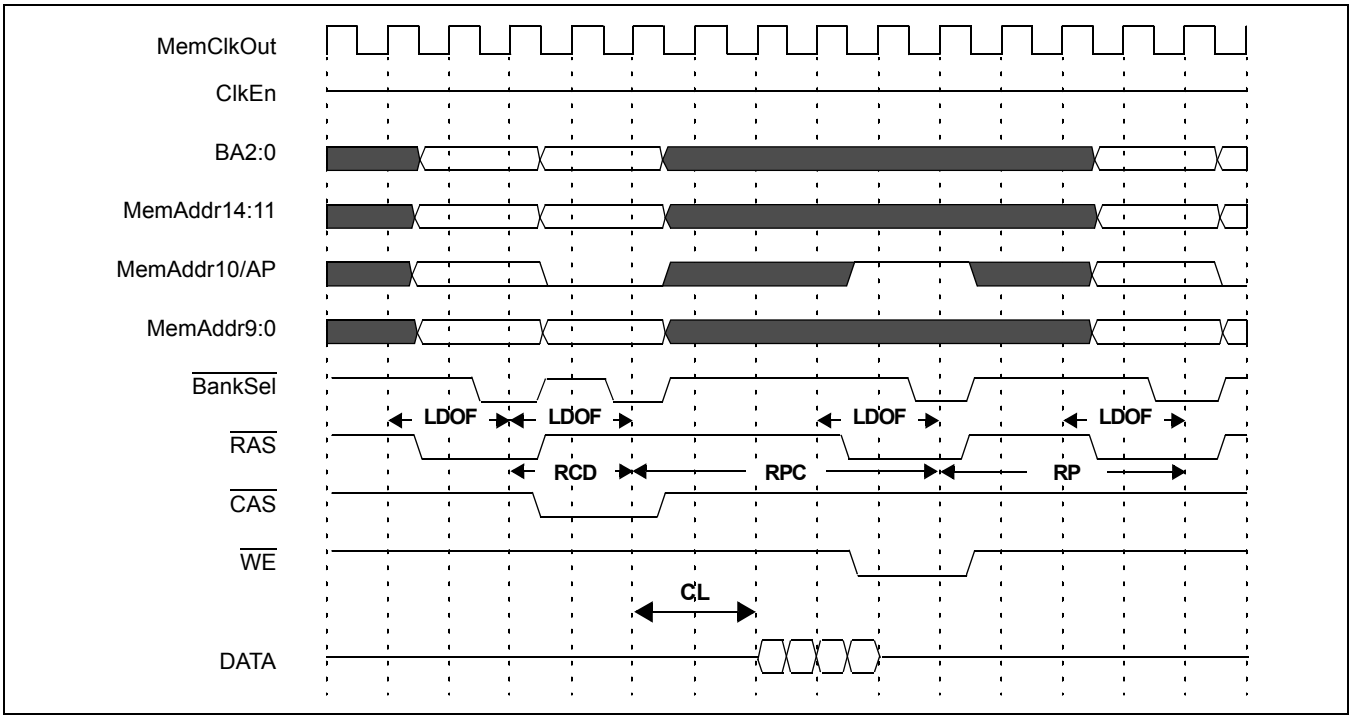


Figure 22-42. Activate - Write - Precharge - Activate

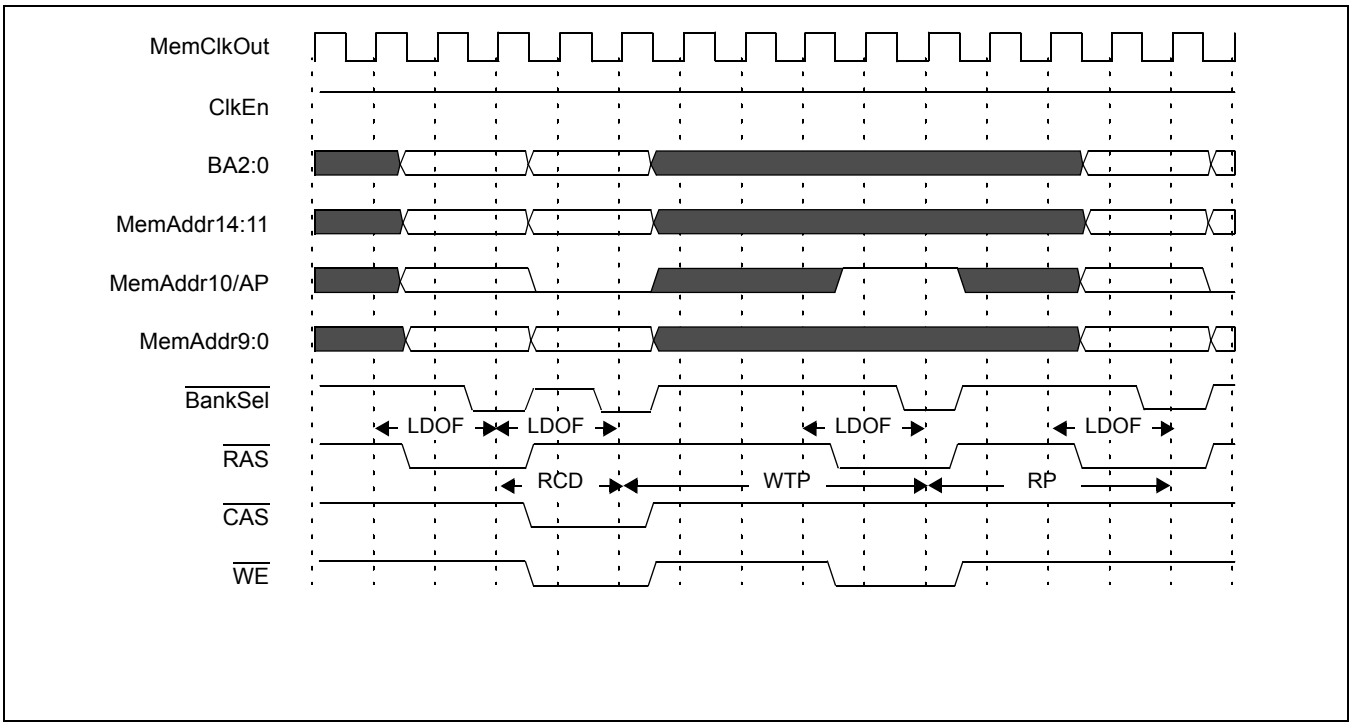
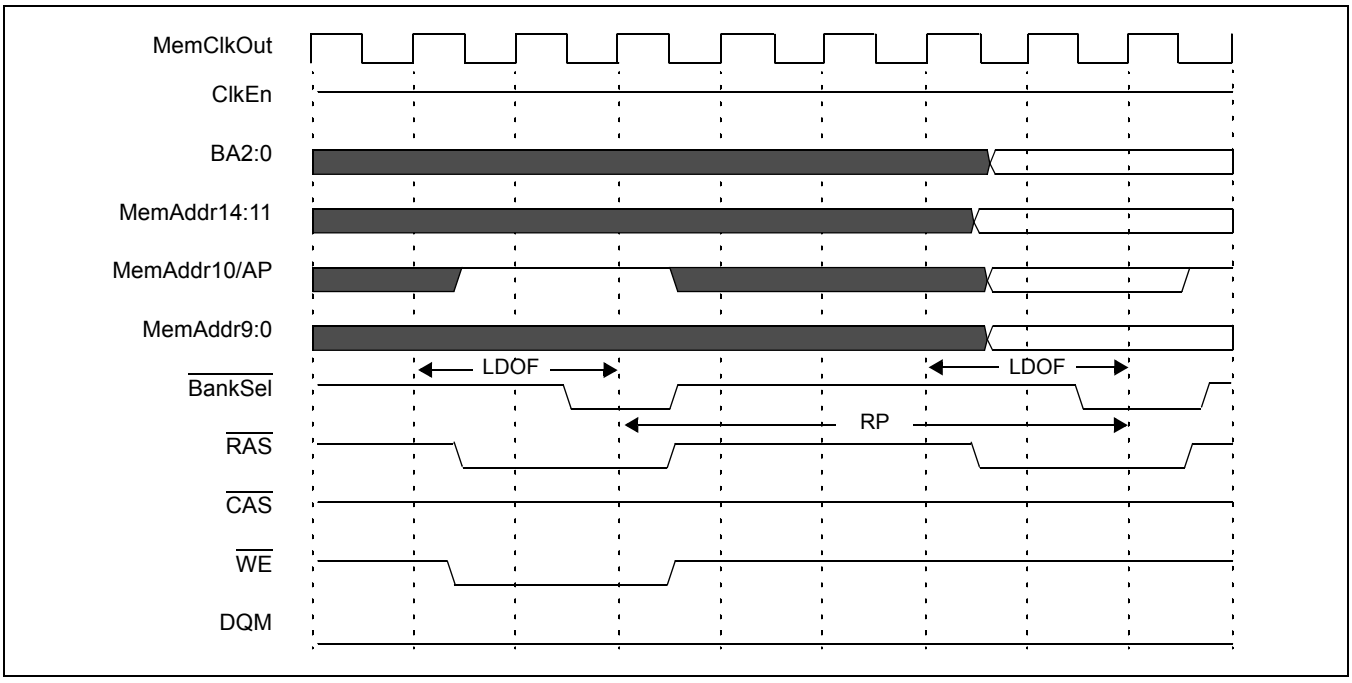


Figure 22-43. Precharge All - Activate



22.2.15.2 Precharge Command

The precharge command instructs the SDRAM to precharge some or all banks, and is generated by the DDR SDRAM controller. The memory address bus contains the command associated with the precharge cycle, which is formatted as shown in *Table 22-25*.

Table 22-25. Precharge Command

Command	BA2	BA1	BA0	MA 14	MA 13	MA 12	MA 11	MA 10/ AP	MA 9	MA 8	MA 7	MA 6	MA 5	MA 4	MA 3	MA 2	MA 1	MA 0
Precharge All Banks								1										
Single Bank Precharge for Chip Select N ($\overline{\text{BankSel}}$)	$\overline{\text{BA2}}$	$\overline{\text{BA1}}$	$\overline{\text{BA0}}$					0										

Note: Shaded entries in this table are "do not care" and are driven with a stable value for every clock cycle.

22.2.15.3 Auto Refresh

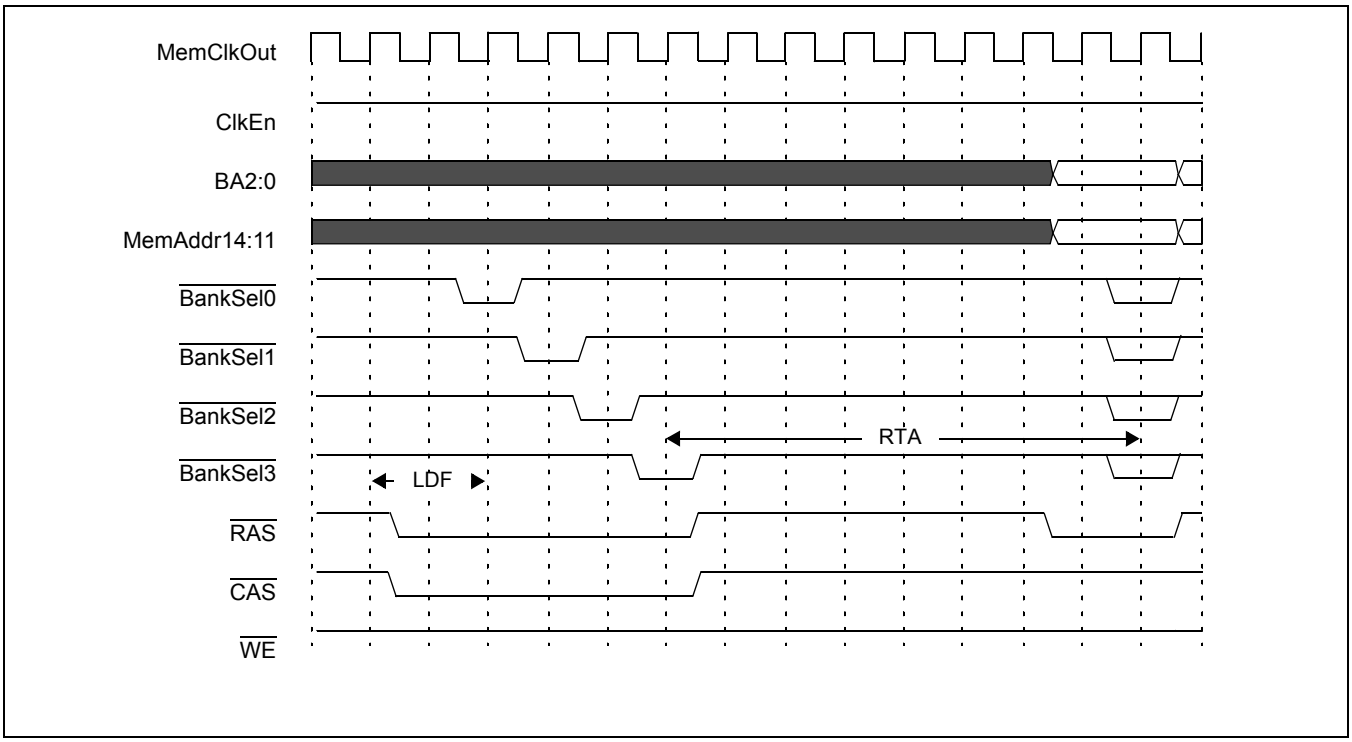
Only ranks enabled in the memory rank configuration register are initialized following reset and refreshed during normal operation. Once the memory controller is enabled and the initialization sequence has completed, the refresh mechanism starts automatically with refreshing of the memory continuing independent of MCIF0_MCOPT2[DCEN].

Refresh requests are generated internally when the refresh timer expires. The refresh interval is programmable via the Refresh Timer Register (MCIF0_RTR). During refresh, all SDRAM accesses are delayed until the current or pending refresh cycle completes.

An option to defer up to four refresh commands is available to improve memory performance. This feature is enabled by setting the DREF bit of MCIF0_MCOPT1. In this mode, memory refresh commands may be deferred until either the command queue is empty or four refresh periods have passed. When these conditions are met, the controller will issue all the deferred refresh commands as a burst without interruption.

Figure 22-44 shows the timing relationships for an automatic refresh.

Figure 22-44. Auto (CBR) Refresh



22.2.15.4 Self-Refresh Operation

The DDR SDRAM controller supports both software- and hardware-initiated self-refresh. The software self-refresh exit trigger mechanism is controlled by resetting (once it has been set) MCIF0_MCOPT2[SREN].

Note: Self-refresh affects only the SDRAM memory.

Although not required, it is recommended that all pending and previously queued requests targeting the DDR SDRAM controller be allowed to complete before starting a software-initiated self-refresh. Any pending or previously queued requests that have not completed before starting a software-initiated self-refresh will stall (including the PLB handshake, if in progress) until self-refresh is exited either by clearing MCIF0_MCOPT2[SREN] or by asserting RESET.

Software-Initiated Self-Refresh Operation

The SDRAM controller supports software-initiated self-refresh for applications that want lower power.

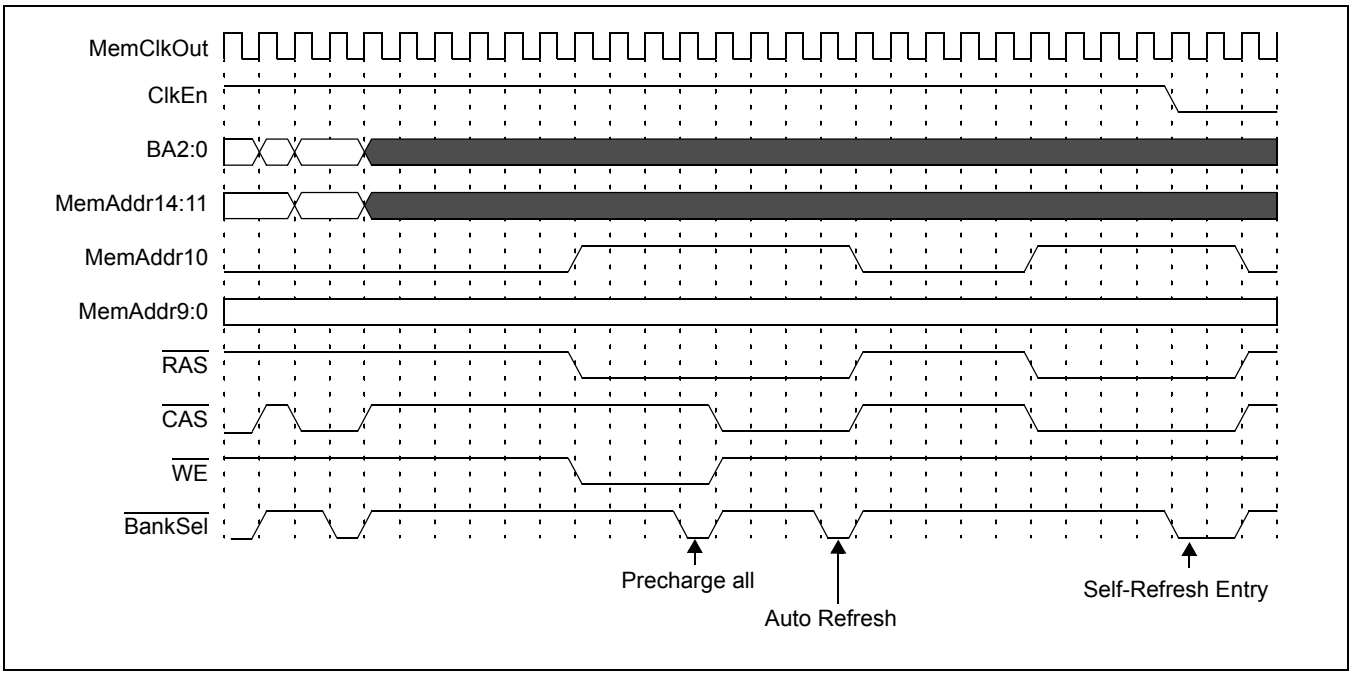
Entry

Software self-refresh is initiated by setting MCIF0_MCOPT2[SREN]. If set, the DDR SDRAM controller:

1. Completes the current request.
2. Performs precharge all to close all open pages.
3. Performs an auto-refresh cycle.
4. Enters self-refresh mode and sets MCIF0_MCOPT2[SRMS].

The DDR SDRAM controller maintains the SDRAM in self-refresh mode, independent of any pending memory access request, until MCIF0_MCOPT2[SREN] is cleared. *Figure 22-45* shows the timing relationships for a self-refresh entry.

Figure 22-45. Self-Refresh Entry



Exit

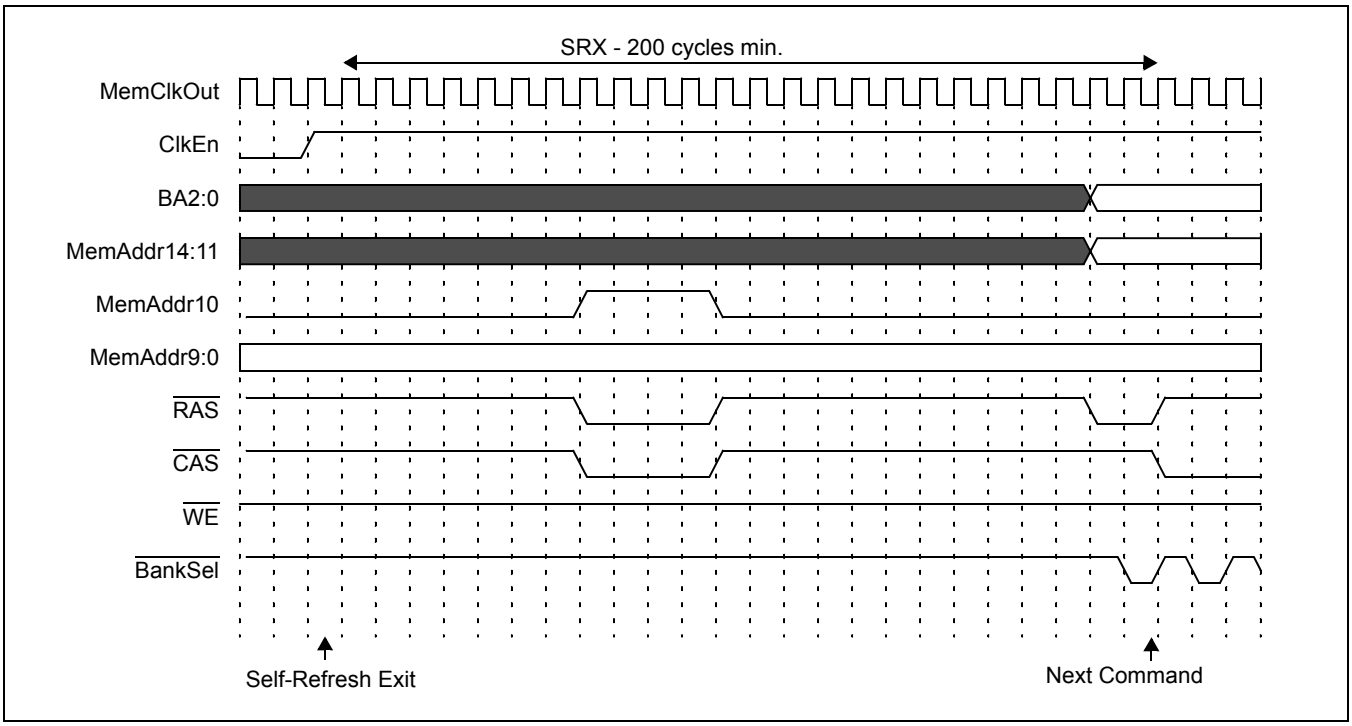
Once MCIF0_MCOPT2[SREN] is cleared, the SDRAM controller:

1. Exits self-refresh mode (CKE = 1).
2. Waits for tXSC cycles for clock to stabilize. If a hardware self-refresh request signal is detected during this wait, then tXSNR is satisfied and self-refresh mode is re-entered (see the next section for a description of this sequence).
3. If MCIF0_MCOPT2[IPTR] was set, the programmed initialization sequence is performed. See the memory specifications for the required self-refresh exit commands.
4. Clears MCIF0_MCSTAT[SRMS].

At this point, software must set MCIF0_MCOPT2[DCEN] = 1. This allows normal memory operations to resume.

Figure 22-46 shows the timing relationships for a self-refresh exit.

Figure 22-46. Self-Refresh Exit



Hardware-Initiated Self-Refresh Operation

The SDRAM controller supports hardware-initiated self-refresh for applications that want lower power.

Entry

Hardware self-refresh is initiated by asserting the HISRRst input. (If the memory controller ClkEn signal is already off, then Steps 1–4 are skipped.)

1. Complete the current memory transaction (if any), including the full data tenure. Pending or queued transactions will not be performed and may be lost.
2. Perform precharge all to close all open pages.
3. Perform any deferred auto refresh cycles and one more auto refresh cycle.
4. Enter self-refresh mode (ClkEn = 0).

Notes:

1. Pending or new memory access request are ignored in this mode.
2. Entering this mode stalls the memory command queues and may hang the processor system.
3. MCIF0_MCSTAT[SRMS] = 1 does not necessarily indicate that the memory is in self refresh mode, but only that a software initiated request has been completed.

Exit

Independent of any pending memory access requests, the SDRAM controller maintains the SDRAM in self-refresh mode, until MCIF0_MCOPT2[SREN] is cleared. Once MCIF0_MCOPT2[SREN] is cleared, the SDRAM controller will perform the following:

1. Exit self-refresh mode (set ClkEn = 1)
2. Wait for tXSC cycles for clock to stabilize. If a hardware self-refresh request signal is detected during this wait, then tXSNR will be satisfied and self-refresh mode will be re-entered (see the next section for a description of this sequence).
3. If MCIF0_MCOPT2[IPTR] was also set, the programmed initialization sequence is performed. See the memory specifications for required self-refresh exit commands.
4. Clear MCIF0_MCSTAT[SRMS] and MCIF0_MCOPT2[SREN]

At this point, software must set MCIF0_MCOPT2[DCEN] = 1 to allow normal memory operations to be accepted.

22.2.16 Registered DIMM Support

Registered interface operation is supported and enabled through MCIF0_MCOPT1[RDEN].

The CAS latency setting must be programmed based solely on the DDR SDRAM device CAS latency. Thus, a DDR SDRAM CAS latency of two clocks corresponds to a DIMM CAS latency of three clocks, and a DDR SDRAM device CAS latency of three clocks corresponds to a DIMM CAS latency of four clocks.

22.2.17 Error Checking and Correction (ECC)

The DDR SDRAM controller supports ECC by enabling MCIF0_MCOPT1[MCHK] (bits 2:3). The ECC circuit provides check bit generation on PLB-to-memory writes and checking/correction on PLB-to-memory reads.

22.2.17.1 32-Bit or 64-Bit SDRAM Interface

The ECC uses either a dual 32-bit or a 64-bit single-bit-error-correction/double-bit-error-detection (SEC/DED) code, depending on the memory interface width. In 32-bit mode, the DDR SDRAM controller implements a dual 32-bit data/8-bit check interface on the memory. In dual 32-bit mode, the ECC (generation, check and correction) is performed on each 32-bit access to the memory, allowing for SEC/DED coverage within each 32-bit word. In 64-bit mode, the DDR SDRAM controller implements a 64-bit data/8-bit check interface on the memory.

22.2.17.2 Read-Modify-Writes

Any single-beat PLB write cycle to memory with byte enables that are not divisible by an ECC word (4-byte and aligned for 32-bit mode, and 8-byte and aligned for 64-bit mode) triggers the ECC block to generate a read cycle to fetch the appropriate quadword, modify that quadword, generate the ECC bits, and write the quadword back to memory.

22.2.17.3 ECC Features

Table 22-26 describes the ECC features.

Table 22-26. ECC Features

Feature	Explanation
Standard SEC/DED coverage	The ECC module corrects all single-bit errors and detects all double-bit errors when reading from memory.
Aligned nibble error detect	The ECC module detects any and all errors which may exist in an aligned 4-bit nibble.
Checking and Correction Disable	ECC error correction may be disabled. Mixed use of ECC and non-ECC ranks is not supported.

22.2.17.4 ECC Timing

Table 22-27 describes the effects of ECC on read and write access timings.

Table 22-27. Effects of ECC on Timing

PLB Transaction	Added Latency	Comment
ECC Enabled: Read	1 Clock	Read Data is registered prior to going through ECC tree.
ECC Enabled: Burst or full single-beat write	None	No extra clock cycles are needed for non-partial writes to perform the ECC.
ECC Enabled: Partial Writes requiring Read-Modify-Write	Time for read	For partial writes, a Read-Modify-Write sequence, including bus turnaround, is required to correctly generate the write check bits and store the resultant data.

22.2.17.5 ECC Configuration Register

The ECC circuit uses MCIF0_MCOPT1[MCHK] to control its operation (see *Memory Controller Options 1 Register (MCIF0_MCOPT1)* on page 766).

The MCHK field allows 3 types of operations of the Memory Controller with the ECC active:

MCHK	Memory ECC	Comment
00	No ECC	
01	ECC Checking and correction with No error reporting	
10	ECC Generation only (No Checking and No correction)	
11	ECC Checking and correction with Error reporting	Normal operation with ECC

22.2.17.6 ECC Error Registers

There are two sets of DCR registers in the Memory Queue and the Memory Controller that records specific ECC errors, and one additional set in the PLB arbitration circuit that could also record these errors if an ECC error causes a timeout on the PLB Bus.

1. Error Status Registers (Mem Queue and Cntl)
- MQ0_ESH Error Status (HB)
- MQ0_ESL Error Status (LL)
- MCIF0_ECCES ECC Error Status

2. Error Address Registers (Mem Queue and Cntl)
- MQ0_EAUH Error Address, Upper 32 Bits (HB)
- MQ0_EALH Error Address, Lower 32 Bits (HB)
- MQ0_EAUL Error Address, Upper 32 Bits (LL)
- MQ0_EALL Error Address, Lower 32 Bits (LL)

3. Error Status Registers on PLB Bus
- PLB4A0_ESRL PLB0_LL Error Status Register Low
- PLB4A0_ESRH PLB0_LL Error Status Register High
- PLB4A1_ESRL PLB1_HB Error Status Register Low
- PLB4A1_ESRH PLB1_HB Error Status Register High

4. Error Address Registers on PLB Bus
- PLB4A0_EARL PLB0_LL Error Address Register Low
- PLB4A0_EARH PLB0_LL Error Address Register High
- PLB4A1_EARL PLB1_HB Error Address Register Low
- PLB4A1_EARH PLB1_HB Error Address Register High

If an ECC error occurs, the ECC Error registers must be reset to enable the recording of any subsequent errors. Reset the registers with a write of 0xFFFFFFFF.

22.2.17.7 Error Detection and Error Handling

The DDR SDRAM controller detects, reports, and logs errors and error status for PLB-to-memory accesses. The controller enables "locking" of the master's Error Status Registers and Error Address Registers for errors reported on the PLB. If locked, the error registers remain locked until software clears the associated master lock error status bits.

An SDRAM ECC correctable or uncorrectable error can activate interrupt signal 4 of the Universal Interrupt controller UIC1[4]. This interrupt remains asserted until the uncorrectable or correctable error bit is cleared in the MCIF0_ECCES register. This level-sensed interrupt may be used to signal the uncorrectable error event to the system.

The error status is recorded in the following Memory Controller registers:

1. MCIF0_ECCES: ECC Error Status Register
2. MQ0_ESH: Error Status (HB)
3. MQ0_ESL: Error Status (LL)

If an ECC error occurs, the ECC error registers must be reset in order to enable subsequent error recording. Perform the reset by writing 0xFFFFFFFF to either MQ0_ESH (or MQ0_ESL) as appropriate.

22.2.17.8 PLB-to-Memory Read and Write Errors

There are two general types of ECC errors: single-bit correctable, and multi-bit uncorrectable. ECC-related errors are detected and logged for DDR SDRAM PLB-to-memory accesses. ECC error checking and correction must be enabled using MCIF0_MCOPT1[MCHK].

If ECC is disabled, or if ECC is enabled and ECC error checking and correction is disabled, no memory access error checking occurs. Specific error types, the information logged, and error responses are detailed in the following sections.

22.2.17.9 Uncorrectable ECC Error on Memory Read

An uncorrectable error detected during a PLB-to-memory read causes the "bad" data being returned from system memory (unchanged) to be transferred on the PLB with the associated ERROR signal. The status and the address associated with the error are logged in the corresponding registers.

The PLB address associated with the error is logged in either PLB4A0_ESRL (for low latency) or PLB4A0_ESRH (for high bandwidth). The error status for the associated master is logged in either PLB4A0_ESRL or PLB4A0_ESRH, and in either MQ0_ESH or MQ0_ESL, depending on the PLB slave access. The ECC Error Status register is updated to indicate that an uncorrectable error occurred with the chip select MCIF0_ECCES[BKnE] associated with the logged error.

The uncorrectable error interrupt signal UIC1[4] is asserted and remains asserted until the uncorrectable error bit is cleared in the MCIF0_ECCES register and in either MQ0_ESL (low latency) or MQ0_ESH (high bandwidth). This level-sensed interrupt may be used to signal the uncorrectable error event to the system.

22.2.17.10 Uncorrectable ECC Error on Memory Partial Write

An uncorrectable error detected on the read portion of a read-modify-write sequence for a PLB-to-memory partial write results in the combining of the data returned from system memory (unchanged) with the "write" data and the writing of it back to memory with new ECC check bits. The associated interrupt signal for PLB slave PLB_MIRQn is asserted to indicate the error. This signal remains asserted until the corresponding bit is cleared in the PLB_MIRQn register. The address associated with the error is logged in either PLB4A0_EARL or PLB4A0_EARH, MQ0_EAUH and EALH pair or the MQ0_EAUL and EALL pair of registers.

The error status for the associated master is logged in either MQ0_ESH or MQ0_ESL, depending on the PLB slave, and in PLB4A0_ESRL or PLB4A0_ESRH. The ECC Error Status register is updated to indicate that an uncorrectable error occurred with the chip select MCIF0_ECCES[BKnE] associated with the logged error. The correctable error interrupt signal (UIC1[4]) is asserted and remains asserted until the Uncorrectable Error bit is cleared in MCIF0_ECCES and in either MQ0_ESL (low latency) or MQ0_ESH (high bandwidth). This level-sensed interrupt may be used to signal the uncorrectable error event to the system.

22.2.17.11 Correctable ECC Error on Memory Read

A correctable error detected on a DDR SDRAM PLB-to-memory read results in the corrected data returned from system memory being transferred to the requesting master device on the PLB. The ECC Error Status register (MCIF0_ECCES) is updated to indicate that a Correctable Error occurred with the byte lane corrected MCIF0_ECCESR[BnCE], check bit status (if applicable), and the chip select MCIF0_ECCES[BKnE] of the logged error.

The correctable error interrupt signal (UIC1[4]) is asserted and remains asserted until the Correctable Error bit is cleared in the MCIF0_ECCES and in either MQ0_ESL (low latency) or MQ0_ESH (high bandwidth). This level-sensed interrupt may be used to signal the Correctable Error event to the system. The address of a single-bit error is not saved. The data that contains the error in the memory system is not corrected in the memory system.

22.2.17.12 Correctable ECC Error on PLB Partial SDRAM Memory Write

A correctable error detected on the read portion of a read-modify-write sequence during a PLB-to-memory partial write (less than a double word) results in the combining of the corrected data returned from system memory with the "write" data and the writing of it back to memory with new ECC check bits.

The ECC Error Status register is updated to indicate that a correctable error occurred with the byte lane corrected MCIF0_ECCES[BnCE], check bit status (if applicable), and the chip select MCIF0_ECCESR[BKnE] of the logged error. The correctable error interrupt signal (UIC1[4]) is asserted and remains asserted until the Correctable Error bit is cleared in the MCIF0_ECCES and in either MQ0_ESL (low latency) or MQ0_ESH (high bandwidth). This level-sensed interrupt may be used to signal the correctable error event to the system.

22.2.17.13 ECC Diagnostics

The occurrence of a correctable single bit error or an uncorrectable multi-bit error can be simulated and the operation of the ECC verified through a series of memory accesses while varying the Memory Data error checking setting (MCIF0_MCOPT1[MCHK]). The general sequence is as follows:

1. Initialize all DDR SDRAM related configuration registers.
2. Enable ECC generation only (MCIF0_MCOPT1[MCHK] = 10).
3. Enable the SDRAM controller (MCIF0_MCOPT2[DCEN] = 1).
4. Initialize (Write) the entire installed memory space to initialize the data and check bits.
5. Allow the DDR SDRAM to become idle.

   To ensure the last write completed, read the last address written. The DDR SDRAM controller maintains read/write order for accesses to the same address.
6. Set the Memory Data error checking to "None" (disable ECC – MCIF0_MCOPT1[MCHK] = 00).
7. Modify one bit (in the case of a CE) or two bits (in the case of a UE) at the address to be tested.
8. Allow the DDR SDRAM to become idle.

   To ensure the last write completed, read the last address written. The DDR SDRAM controller maintains read/write order for accesses to the same address.
9. Set the Memory Data error checking to "ECC Checking and Correction."
10. Read the corresponding address.

In the case of simulating a Correctable Error, the corrected data should be returned to the system. In the case of simulating an Uncorrectable Error, the original data (with the two changed bits) should be returned to the system. In either case, the described error response corresponding to the error simulated should also occur.

If multiple errors occur, only the first error is latched. Subsequent errors are not recorded in the error registers until after the firmware clears the latched error by either clearing MCIF0_ECCES or by forcing a system reset.

22.2.17.14 ECC Code Matrix

Table 22-28 through *Table 22-32* indicate which data bits are exclusive-or'ed together in order to generate the ECC check bits. The data bits included in the bit-wise exclusive-or are marked with an X. Each row represents an equation for a particular check bit. For example, row 0 contains the equation for ECC0.

Table 22-28 and *Table 22-29* are used for the data at even word addresses or word 0. Address bit 29 is 0 for even 32-bit word addresses. These are addresses ending in 0xXXXXXXX0 or 0xXXXXXXX8.

Table 22-30 and *Table 22-31* are used for the data at odd word addresses or word 1. Address bit 29 is 1 for odd 32-bit word addresses. These are word addresses ending in 0xXXXXXXX4 or 0xXXXXXXXC.

In 64-bit mode, the DDR controller generates a particular ECC bit by an exclusive-or of the results from *Table 22-28* and *Table 22-29* with the results of *Table 22-30* and *Table 22-31* as indicated in *Table 22-32*.

Example for ECC0 in 64-bit mode:

UC0 = D0⊕D1⊕D2⊕D3⊕D4⊕D5⊕D6⊕D7⊕D8⊕D9⊕D10⊕D11⊕D12⊕D13⊕D14⊕D15⊕D19⊕D23⊕D27⊕D31

LC0 = D34⊕D38⊕D42⊕D46⊕D51⊕D55⊕D59⊕D61

ECC0 = UC0⊕LC0

In 32-bit mode, the DDR controller generates a particular ECC bit using the results from either Tables 23-23 and 23-24 or Tables 23-25 and 23-26 depending on the address.

Example for ECC0 in 32-bit mode:

UC0 = D0⊕D1⊕D2⊕D3⊕D4⊕D5⊕D6⊕D7⊕D8⊕D9⊕D10⊕D11⊕D12⊕D13⊕D14⊕D15⊕D19⊕D23⊕D27⊕D31

LC0 = D34⊕D38⊕D42⊕D46⊕D51⊕D55⊕D59⊕D61

ECC0 = UC0 if the data is located at a 32-bit word address ending in 0xXXXXXXX0 or 0xXXXXXXX8.

ECC0 = LC0 if the data is located at a 32-bit word address ending in 0xXXXXXXX4 or 0xXXXXXXXC.

Table 22-28. 32-Bit Mode ECC Code Matrix for Word 0 (Part 1 of 2)

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X	
X				X				X				X				X	X	X	X	X
	X				X				X				X			X				X
		X				X				X				X			X			
			X				X				X				X			X	X	
				X	X	X	X					X	X	X	X					X
								X	X	X	X	X	X	X	X					
X	X	X	X									X	X	X	X	X	X	X		

Table 22-29. 32-Bit Mode ECC Code Matrix for Word 0 (Part 2 of 2)

21	22	23	24	25	26	27	28	29	30	31	UC0	UC1	UC2	UC3	UC4	UC5	UC6	UC7
		X				X				X	X							
X	X	X	X	X	X	X	X	X	X	X		X						
			X				X						X					
X				X				X						X				
	X	X			X	X			X	X					X			
X	X	X					X	X	X	X						X		
			X	X	X	X	X	X	X	X							X	
		X				X	X	X	X									X

Table 22-30 and *Table 22-31* show the 32-bit mode ECC code matrix for word 1.

Table 22-30. 32-Bit Mode ECC Code Matrix for Word 1 (Part 1 of 2)

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
		X				X				X				X					X
			X				X				X				X	X			
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X		X		
X				X				X				X						X	
	X	X	X		X	X	X		X	X	X		X	X	X				
				X	X	X	X					X	X	X	X	X	X	X	X
								X	X	X	X	X	X	X	X	X	X	X	X
X	X					X	X			X	X	X	X						

Table 22-31. 32-Bit Mode ECC Code Matrix for Word 1 (Part 2 of 2)

52	53	54	55	56	57	58	59	60	61	62	63	LC0	LC1	LC2	LC3	LC4	LC5	LC6	LC7
			X				X		X			X							
X				X						X			X						
	X				X						X			X					
		X				X		X	X	X	X				X				
								X	X	X	X					X			
X	X	X	X														X		
				X	X	X	X	X	X	X	X							X	
X	X	X	X	X	X	X	X		X	X	X								X

Table 22-32 shows the 64-bit mode ECC code matrix. It is derived from the upper (UC 0:7) and lower (LC 0:7) check bits calculated for 32-bit mode.

Table 22-32. 64-Bit Mode ECC Code Matrix

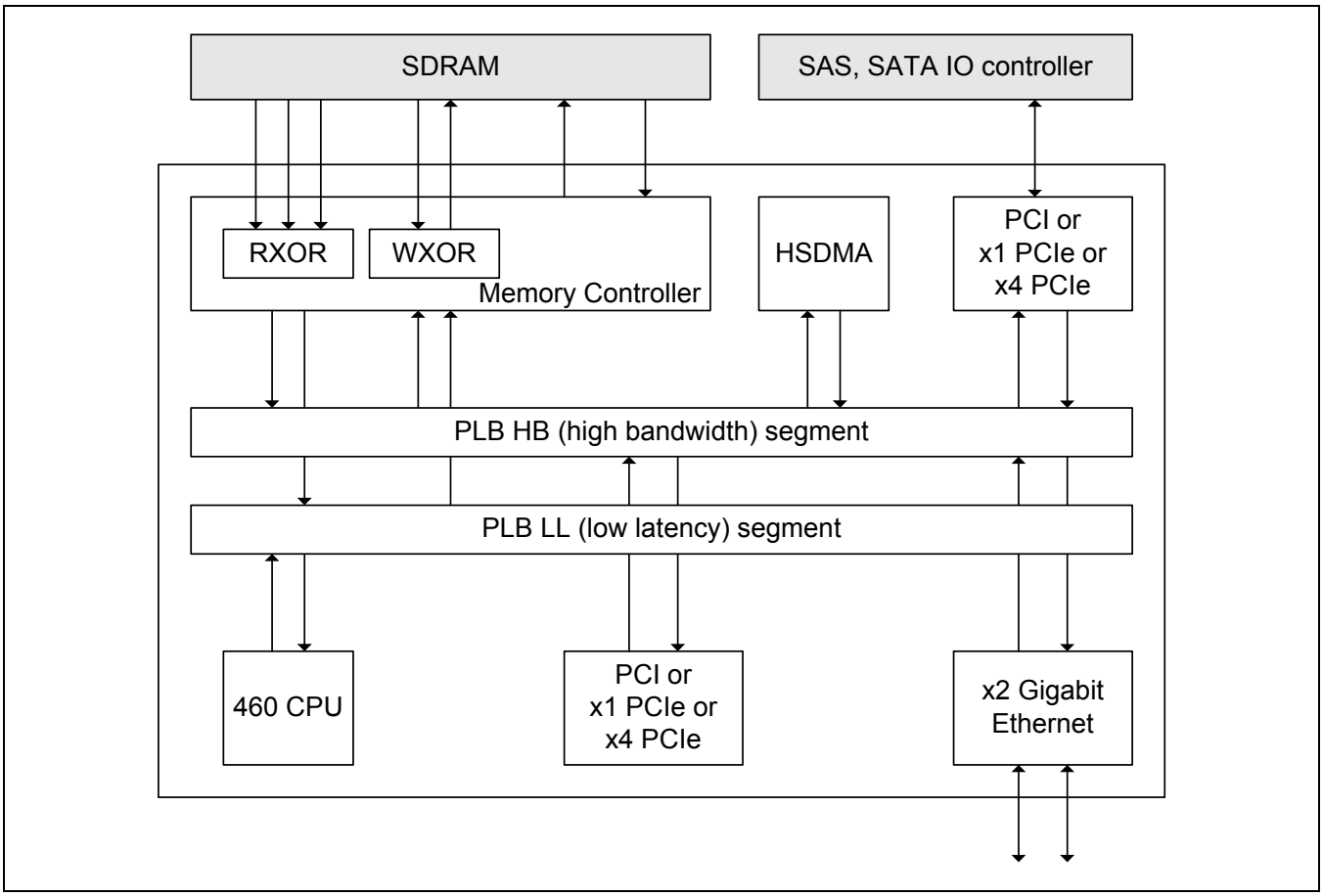
UC0	UC1	UC2	UC3	UC4	UC5	UC6	UC7	LC0	LC1	LC2	LC3	LC4	LC5	LC6	LC7	C0	C1	C2	C3	C4	C5	C6	C7
X								X								X							
	X								X								X						
		X								X								X					
			X								X								X				
				X								X								X			
					X								X								X		
						X								X								X	
							X								X								X

23. RAID5 (PPC460EX/EXr Only)

The PPC460EX/EXr supports RAID5 functionality. This section describes the circuitry and programming model for RAID5 support hardware implemented in the PPC460EX/EXr processor. RAID5 hardware acceleration is implemented in the Memory Controller attached to the internal PLB (Processor Local Bus) bus.

Figure 23-1 shows the PPC460EX/EXr the paths for RAID applications with high data throughput. The subsequent sections describe the operation of the XOR logic inside the Memory Controller.

Figure 23-1. PPC460EX/EXr Block Diagram in a RAID Application



The PPC460EX/EXr has two internal busses: HB (high bandwidth) and LL (low latency). The LL bus is typically used for moving control information. The CPU is connected to the LL bus. The HB bus is typically used for moving data. For example, the HSDMA (high speed DMA) is connected to the HB bus. The memory controller is connected to both busses. While the CPU is writing descriptors, the DMA engine can move data as prescribed by these descriptors. Each bus has a read segment and a write segment. Thus, four transactions can be happening at the same time. The PLB has a 64-bit address. One of the upper 30 bits determines if the memory is accessed via HB or LL (*Figure 23-3*). The CPU has a 32-bit address. The TLB (translation look-ahead buffer) table creates 64-bit addresses out of 32-bit CPU addresses. The CPU writes 64-bit addresses to the descriptors in two 32-bit chunks of data.

In RAID5, an XOR is needed to compute parity and recover data in the case of a disk failure. For example, $P = A_0 \oplus A_1 \oplus A_2$ where A_i denotes a data block on a single disk drive. This operation can be performed in the memory controller as HSDMA writes to the SDRAM with cues in the upper 32-bit PLB addresses in the DMA descriptor.

23.1 Using XOR with HSDMA or an External DMA

The PPC460EX/EXr implements XOR hardware in the memory queue to accelerate Parity (P) computations for RAID5. The XOR logic is activated if a read or write access to SDRAM is decoded as a transaction with selection of the XOR. This is accomplished by adding cues to the upper 28 bits of source or destination address. Note that the device can drive up to 16 GB of memory using the lower 34 bits of address. The format of the cues is shown in *Figure 23-2*. Cueing the source address activates the RXOR circuitry in the memory controller. Cueing the destination address activates the WXOR circuitry in the memory controller.

The read or write operation that uses XOR hardware assist can be performed by any master on the PLB bus as long as it can generate the upper 28 bits of the read or write address. This operation is valid for access to the memory through the High Bandwidth (HB) PLB slave bus.

Figure 23-2 shows the format for SDRAM addresses for XOR operations on the HB PLB bus.

Figure 23-2. Format of SDRAM Address for XOR Operation on the HB PLB Bus

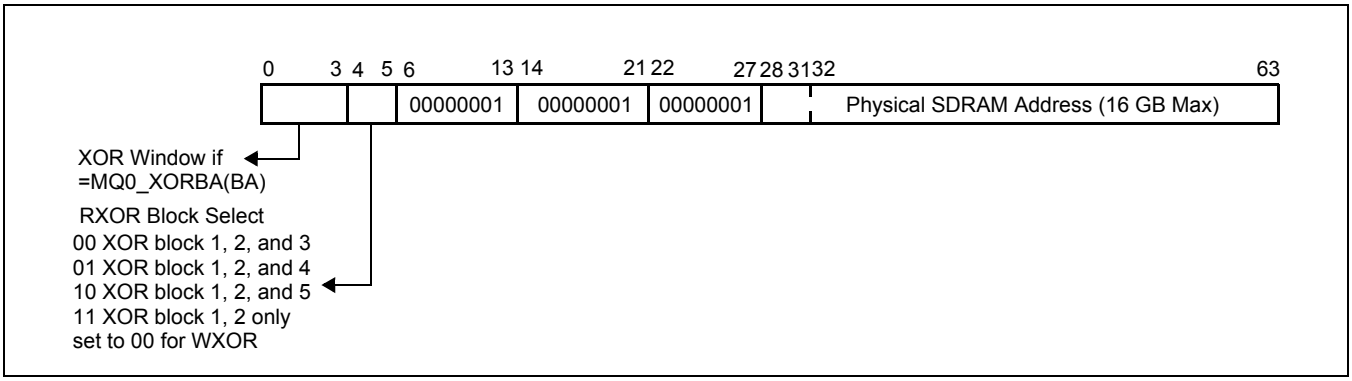


Figure 23-3 shows the format for SDRAM addresses for normal operation.

Figure 23-3. Format of SDRAM Address for Normal Operation on the HB or LL PLB Bus

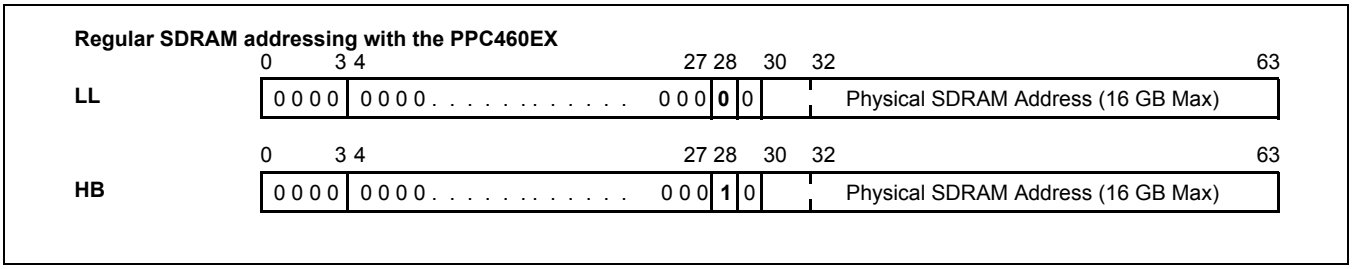
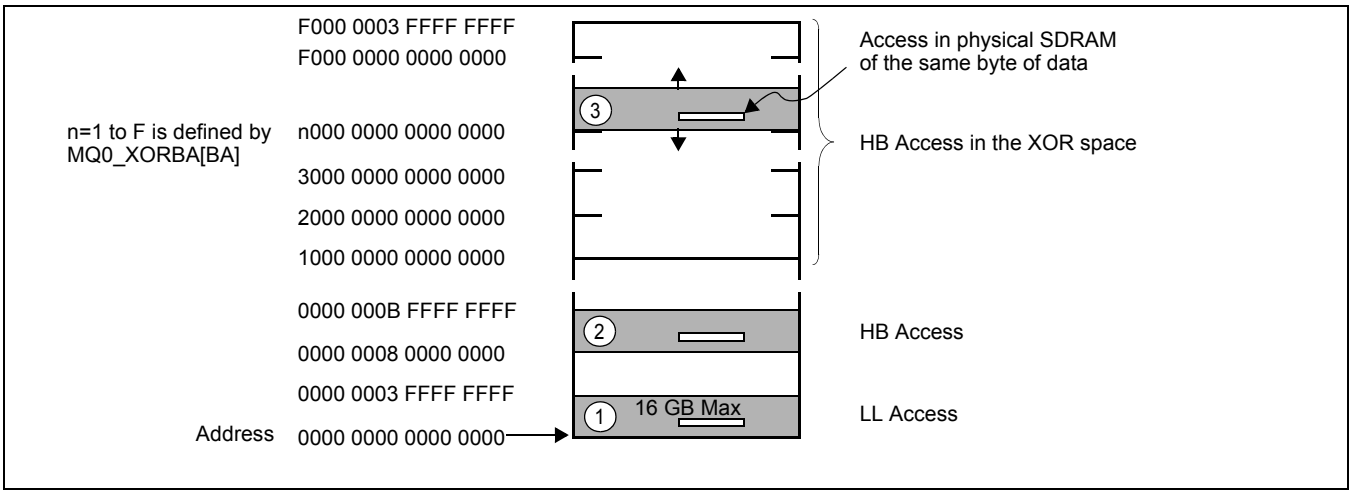


Figure 23-4 shows PPC460EX/EXr memory mapping in PLB space. As shown in the figure, access of the same byte in PPC460EX/EXr memory (SDRAM up to 16 GB) can be accomplished in three separate spaces on the two PLB slave buses.

- 1 - LL: Standard memory access address range: 0 0000 0000 - 3 FFFF FFFF
- 2 - HB: Without XOR, address range: 8 0000 0000 - B FFFF FFFF
- 3 - HB: With XOR, address range: n000 0000 0000 0000 - n000 0003 FFFF FFFF where n ranges from 1 to F and is equal to MQ0_XORBA(BA) field

Figure 23-4. PPC460EX/EXr Local SDRAM Memory Mapping in PLB Space



The activation of Read XOR (RXOR) and Write XOR (WXOR) is done through the programming of the CDBs (Command Descriptor Block) of the HSDMA controller. An external DMA may also be used. The CDBs do not contain XOR opcodes. The source (for the RXOR method) and destination (for the WXOR method) addresses may be programmed as described previously to activate XOR functionality.

For RXOR, the source data must be in properly spaced blocks in the 460EX/EXr local memory. A single cued local SDRAM source address is sufficient to cause XOR computation of 3 properly spaced blocks. These blocks may be contiguous as in the case of regions 1, 2, and 3. The other options are 1,2,4 and 1,2,5 and 1,2. Blocks 1,2,3,4,5 are contiguous. See *Figure 23-6*. The result of the computation may be placed at the destination address in the DMA CDB if the DMA is internal. An external DMA reading out of the 460EX/EXr local memory need not have a destination address in this PCI space or a destination address at all. It may send the result to the link. It is also possible to use two destination addresses (multicast) with HSDMA.

For WXOR, one needs to use the same 460EX/EXr local SDRAM destination address in multiple DMA CDBs. Each CDB creates an XOR of what was previously in the destination address with what is in the source and writes the result back at the destination. For example with an 8-drive RAID5 configuration there will be 7 CDBs each referring to a different source address but all referring to the same destination address which is the local memory address for parity for this stripe. An external DMA writing to the 460EX/EXr local memory need not have a source address in this PCI space or a source address at all. It may fetch the data from the link.

Table 23-1. Memory Space Address Used for RAID Operation Through RXOR or WXOR With Internal DMA

Operation	DMA Source Address	DMA Destination Address
WXOR	a memory address if any	to local SDRAM with RAID hardware assist
RXOR	from local SDRAM with RAID hardware assist	a memory address if any

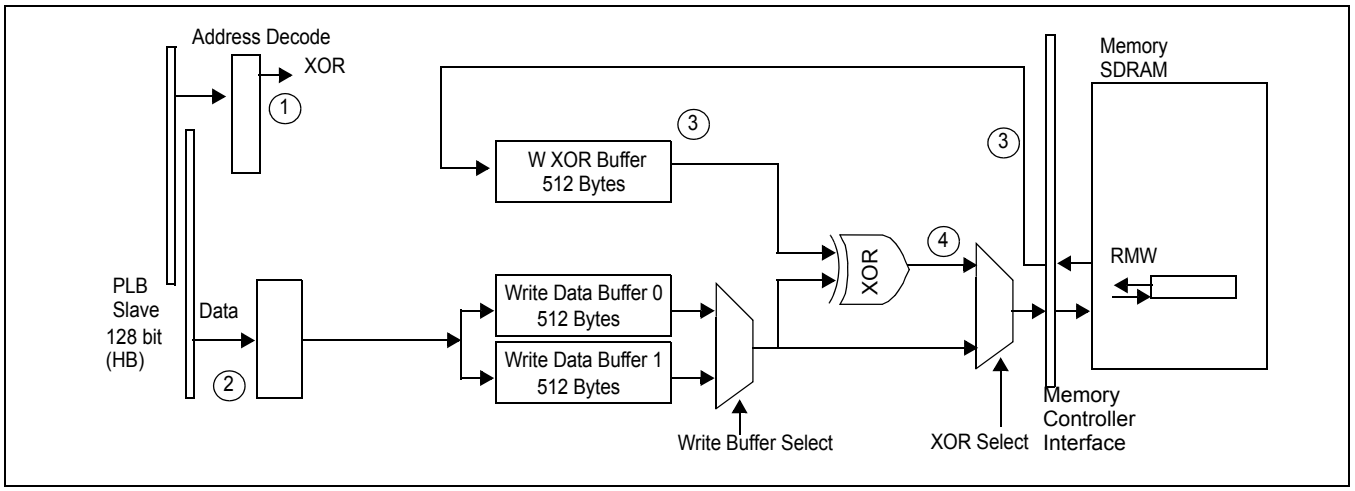
23.1.1 Write XOR

The Write XOR (WXOR) operation is basically a Read-Modify-Write operation, with an XOR of data from memory (current data) and data from the PLB (incoming data). Here are the steps (circled numbers in *Figure 23-5*):

1. Write operation presented by a PLB Master decoded by the Memory controller.
2. Incoming data (at the source CDB is there is a local source) is stored into one of the two write data buffers.
3. Current data (at the destination address) is read from memory and written into the WXOR buffer.
4. Data from the buffers in steps 2 and 3 are XORed and written into memory at the same address as data in step 3 (the destination address).

Figure 23-5 shows the logic for writing data with XOR.

Figure 23-5. Write Data with XOR Logic



Note that the address of the "current data" in step 3 is the destination address in the DMA command descriptor block. This address is also used in step 4. The "incoming data" in step 2 is the source. Its address may be the DMA descriptor source address if the DMA was internal. If the DMA is external, this data comes via a PCI port.

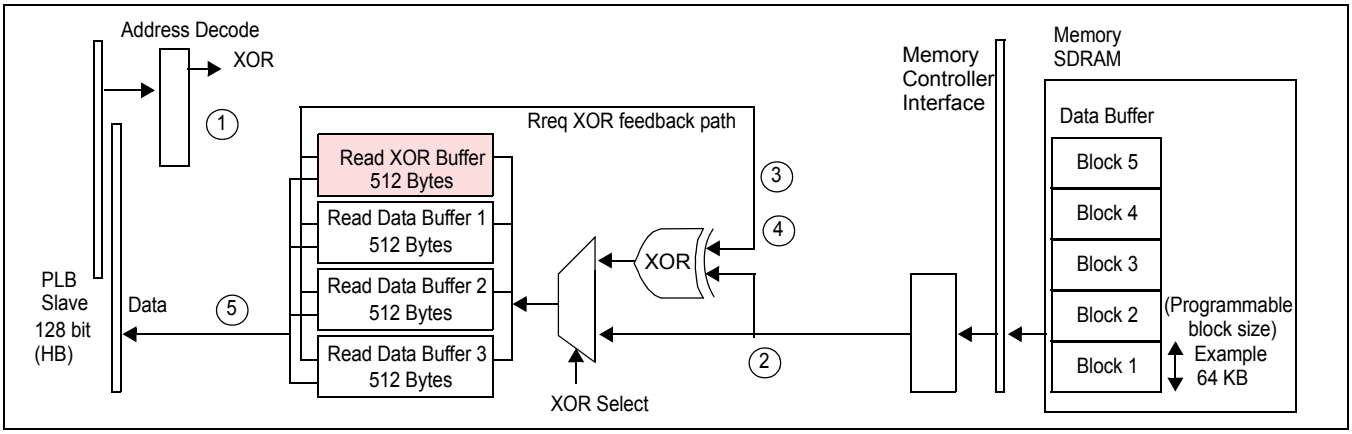
23.1.2 Read XOR

The read XOR (RXOR) operation is performed in response to a memory read to the XOR address region of the internal PLB bus address space. The response is to XOR three data buffers:

1. Read operation presented by a PLB Master decoded by the Memory controller.
2. Read data out of the source address (block 1) and place it in the *Read XOR Buffer*.
3. Read data out of the source address (block 2 = block 1 + block_size), XOR with current contents of the *Read XOR Buffer*, and write back to the *Read XOR Buffer*.
4. Continue if needed with one of the blocks 3, 4, or 5.
5. When the original Read Request (Rreq) comes back, return the data in the *Read XOR Buffer* to the PLB master.

Figure 23-6 shows the logic for reading data with XOR.

Figure 23-6. Read Data with XOR Logic



Note that the address of the "read data" is the source address in the DMA command descriptor block. The source address must point to local SDRAM attached to the 460EX/EXr DRAM controller port.

23.2 Memory Queue Module Registers

The memory queue registers listed in the following table are accessed by means of the DCR bus.

Only the following memory queue registers are used for XOR operations:

- MQ0_XORBA XOR Base Address (HB)
- MQ0_CF2H Configuration 2 (HB)
- MQ0_BAUH PLB Base Address, Upper 32 Bits (HB)
- MQ0_CFBHL Configuration Between HB and LL Paths

Note: No register is used to determine whether the RXOR or the WXOR is to be used. That is determined solely by decoding the address on the PLB with the Read/Write command.

Table 23-2. Memory Queue Register Map

Mnemonic	Register	DCR Address	Access	Page
MQ0_B0BAS	Bank 0 Base Address and Size	0x0040	R/W	N/A
MQ0_B1BAS	Bank 1 Base Address and Size	0x0041	R/W	N/A
MQ0_B2BAS	Bank 2 Base Address and Size	0x0042	R/W	N/A
MQ0_B3BAS	Bank 3 Base Address and Size	0x0043	R/W	N/A
MQ0_XORBA	XOR Base Address (HB)	0x0044	R/W	821
MQ0_CF1H	Configuration 1 (HB)	0x0045	R/W	N/A
MQ0_CF2H	Configuration 2 (HB)	0x0046	R/W	821
MQ0_ESH	Error Status (HB)	0x0047	R/W	N/A
MQ0_EAUH	Error Address, Upper 32 Bits (HB)	0x0048	R	N/A
MQ0_EALH	Error Address, Lower 32 Bits (HB)	0x0049	R	N/A
MQ0_BAUL	PLB Base Address, Upper 32 Bits (LL)	0x004A	R/W	821
MQ0_CF1L	Configuration 1 (LL)	0x004B	R/W	N/A
MQ0_ESL	Error Status (LL)	0x004C	R/W	N/A
MQ0_EAUL	Error Address, Upper 32 Bits (LL)	0x004D	R	N/A
MQ0_EALL	Error Address, Lower 32 Bits (LL)	0x004E	R	N/A
MQ0_CFBHL	Configuration Between HB and LL Paths	0x004F	R/W	822
MQ0_BAUH	PLB Base Address, Upper 32 Bits (HB)	0x0050	R/W	821

23.2.1 XOR Base Address Register (MQ0_XORBA)

This register is used to select memory controller based XOR hardware assist for RAID applications.Any non-zero value will do. The source or destination address that is intended to be cued must have its upper four bits match this value. This register is typically set during initialization.

Figure 23-7. XOR Base Address Register (MQ0_XORBA)

0:3	BA	XOR Base Address	Setting base address [BA] bits to zero disables the XOR engines in the memory controller. If one or more of these bits is non-zero and if the upper four bits of the address match this value then the memory controller based XOR engines (RXOR if source address, WXOR if destination address) are activated.
4:31		Reserved	

23.2.2 Configuration Register 2 (MQ0_CF2H)

Figure 23-8. Configuration Register 2 (MQ0_CF2H)

0:22	BS	Read XOR Block Size	Size of a block in a Read XOR set. The Read XOR block size is in 512-byte increments. See *Figures 2* and *6*. The block size is typically equal to the strip (also called chunk) size in a RAID set. For example if chunk size is 256KB then set these bits to 512.
23:31		Reserved	

23.2.3 PLB Base Address Register Upper 32 Bits (LL) (MQ0_BAUL)

MQ0_BAUL contains the 32 most significant bits of the PLB base address for the Low Latency (LL) path between the PLB and memory. If the contents of this register match the PLB address, access to the SDRAM memory is done on the LL bus. This register should be set to 0x0000 0000.

Figure 23-9. PLB Base Address Register Upper 32 Bits (LL) (MQ0_BAUL)

0:31	BA	Upper 32 bits of PLB Base Address	This address is compared to the upper 32 bits of the PLB address. 0x 0000 0000 must be set for Low Latency (LL) PLB access.

23.2.4 PLB Base Address Register Upper 32 Bits (HB) (MQ0_BAUH)

MQ0_BAUH contains the 32 most significant bits of the PLB base address for the High Bandwidth (HB) path between the PLB and memory. If the contents of this register match the PLB address, access to the SDRAM memory is done on the HB bus. This register should be set to 0x0000 0008.

Figure 23-10. PLB Base Address Register Upper 32 Bits (HB) (MQ0_BAUH)

0:31	BA	Upper 32 bits of PLB Base Address	Each SDRAM memory byte of the 16-GB space is accessible either on the Low Latency (LL) or the High Bandwidth (HB) PLB slave. 0x 0000 0008 must be set if High Bandwidth (HB) PLB access is required.

23.2.5 Configuration Between HB and LL Paths Register (MQ0_CFBHL)

MQ0_CFBHL contains information about transactions that involve both the High Bandwidth (HB) path and the Low Latency (LL) path between the PLB and memory.

Figure 23-11. Configuration Between HB and LL Paths Register (MQ0_CFBHL)

0:3	TPLM	Transaction Passing Limit	Indicates the maximum number of consecutive times a transaction (read or write) in the Low Latency (LL) path is allowed to pass eligible transactions in the High Bandwidth (HB) path. Default is 1 time. A value of zero indicates that there is no limit on the number of times an LL transaction can pass HB transactions. Bit 4 must be enabled for this value to be significant.
4	TPEN	Transaction Passing Enable	Enables transaction passing feature.
5	MQST	MQ Stop	Allows the MQ to be *halted* from issuing commands on MCIF. Enqueuing is still allowed on PLB. *For Test and Lab use only*. Allows testing of specific configurations of enqueued requests.
6:8	HBCL	MCIF Cycle Limit	When an HB request has been passed, indicates the maximum number of consecutive MCIF (Memory Controller) cycles for the HB request that will be run before allowing another LL request.
9:30		Reserved	
31	H2OMW	H2O MQ Watermark	Used for revision control in releases of netlists to AppliedMicro.

24. Direct Memory Access Controller

The Direct Memory Access (DMA2P40) controller is a Processor Local Bus (PLB) and On-chip Peripheral Bus (OPB) master which supports the autonomous transfer of data between memory and peripherals and from memory-to-memory. The controller provides four DMA channels, each of which has an independent set of configuration registers. Each channel has its own control, count and control, source address, destination address, and scatter/gather address registers. Once these registers are programmed, the DMA controller performs the requested data transfer without the need for processor intervention.

The four DMA channels also support scatter/gather transfers. During a scatter/gather transfer, the configuration registers for a particular DMA channel are automatically loaded from a data structure in memory instead of being individually programmed. Since the scatter/gather address register is updated in this process, the channel can optionally reconfigure itself for another transfer when the current one completes.

As PLB and OPB master, the DMA can read and write any address accessible by the processor. This includes memory and memory-mapped peripherals on the external bus controller (EBC) interface and SDRAM memory and PCI/PCI Express addresses that have been mapped into PLB address space. The DMA controller can also service DMA peripherals attached to the EBC by means of the DMAReqn, DMAAckn, and DMAEOTn I/Os, along with the OPB-attached UART0:3.

24.1 External Interface Signals

Figure 24-1 illustrates the external I/Os associated with the DMA controller, and *Figure 24-2* illustrates the EBC I/Os used during external peripheral transfers. External peripheral and EBC device-paced memory transfers request service from the DMA controller by driving a DMA request line (DMAReq0:3) active. For peripheral mode transfers the DMA controller acknowledges the request and transfers data by asserting a DMA acknowledge signal (DMAAck0:3). In contrast, an EBC device-paced memory-to-memory transfer occurs when the chip select (PerCSn) associated with the memory location is driven active. The timing of PerCSn and the other EBC I/Os is determined by the Bank Access Parameter Register (EBC0_BnAP) for the particular memory location. *Table 24-1* describes the DMA external I/O usage.

Table 24-1. DMA Controller External I/Os

Signal	Usage
DMAReqn	DMA request signal used to request either a peripheral mode transfer or an EBC device-paced memory-to-memory transfer.
DMAAckn	DMA acknowledge signal which instructs an EBC-attached DMA peripheral to transfer data.
DMAEOTn	End of Transfer or Terminal Count signal used to stop the channel when programmed as EOT. When configured as TC, goes active when the channel transfer count register (DMA2P40_CTCn) reaches zero.

Note: The active level (polarity) of DMAReqn, DMAAckn, and DMAEOTn are individually programmable for external signals only. All other signals are active high.

Figure 24-1. DMA Controller External Bus Control Signals

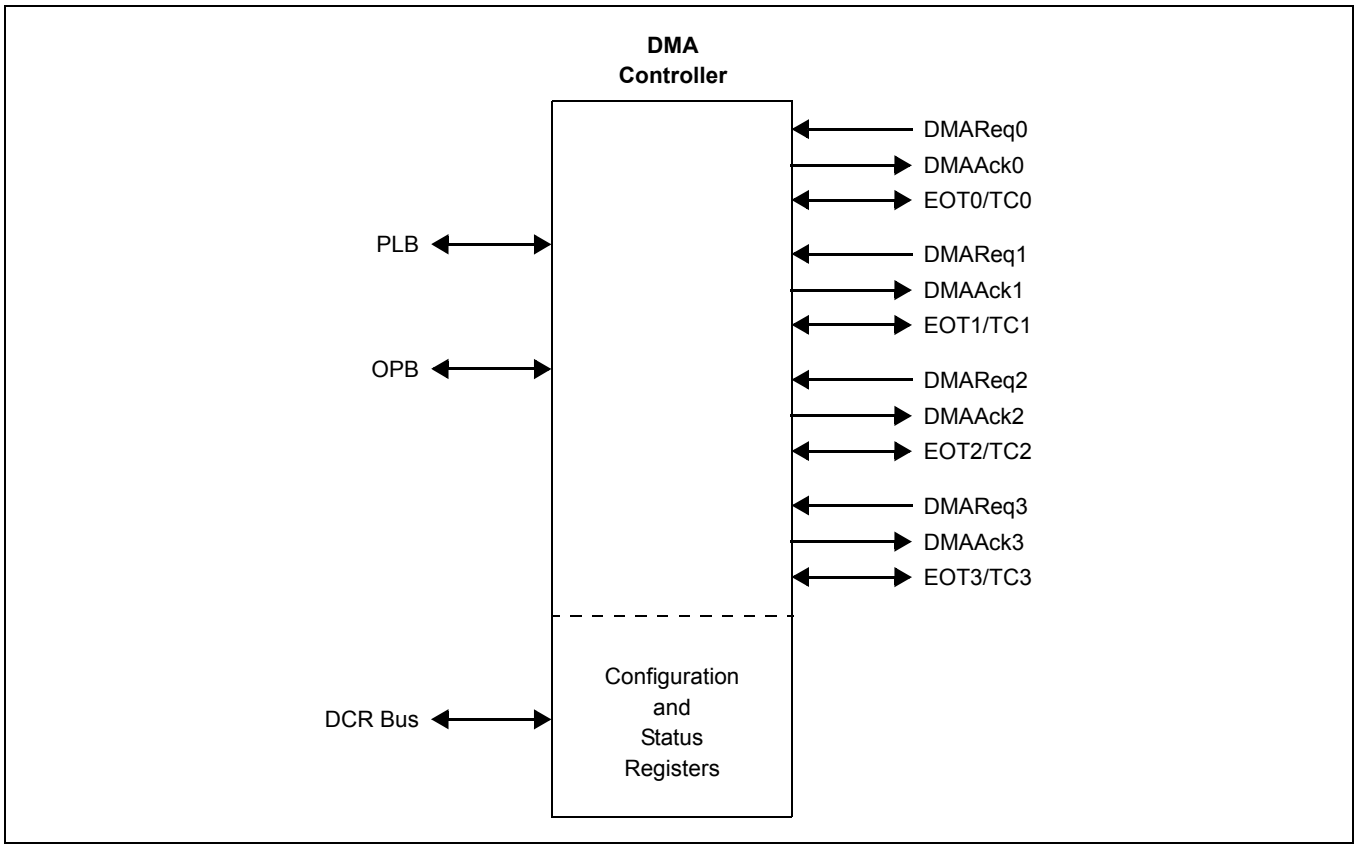
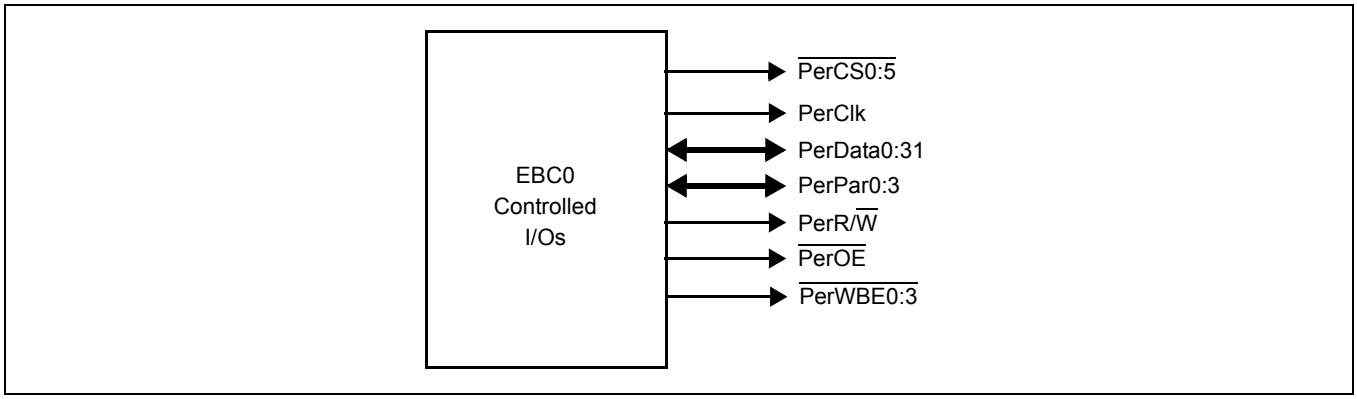


Figure 24-2. External Bus Control Signals



24.2 DMA Transfers

As a specialized controller, the DMA provides system designers and programmers with a highly efficient method of moving data. During any DMA transfer the controller always buffers data read from the source prior to writing the data to the destination. Since many buses, including the internal PLB, provide substantially better performance when bursting data, the DMA controller includes a 128-byte (8-quadword) buffer. This buffer is enabled on a per-channel basis by setting DMA2P40_CRn[BEN] and serves to minimize the number of discrete memory transactions. Each of the four DMA channels is configurable for either peripheral or memory-to-memory transfers.

24.2.1 Peripheral Mode Transfers

Peripherals are either devices attached to the EBC interface via the DMAReqn and DMAAckn lines, or the internal serial ports (UART0:3). Memory is any address accessible from the PLB or OPB, including PLB-mapped PCI/PCI Express address space, SDRAM memory, and external memory. During a peripheral mode transfer the peripheral requests a DMA transfer by asserting a DMA request line. For UART0:3, this signal is internal to the chip, while external peripherals use one of the DMAReqn lines. When the requesting channel has the highest priority of any active channel, the DMA executes a peripheral transfer, and the requesting device receives a DMA acknowledge signal. In the case of external peripherals, the appropriate DMAAckn line is driven to the active state, with the timing specified in the DMA Control Register for the channel (DMA2P40_CRn).

There are two types of peripheral mode transfers: peripheral-to-memory and memory-to-peripheral. A peripheral-to-memory transfer reads data from a DMA device, while a memory-to-peripheral transfer writes data. In both cases, the peripheral interface transfers data at the programmed width of the peripheral (DMA2P40_CRn[PW]). When the DMA buffer is disabled for the active channel (DMA2P40_CRn[BEN] = 0), each peripheral transfer causes a corresponding memory operation.

When buffering is enabled during a peripheral-to-memory transfer, data is collected until the 128-byte buffer is full, the peripheral deasserts DMAReqn, or the channel completes. The buffer contents are then written to the target memory as efficiently as possible.

If the destination address is located on PLB space, the data is written to memory with one burst operation if the destination address is quadword aligned and the buffer is full; or, it is written with a combination of single beat and burst transfers, depending on the destination address alignment and the amount of data stored in the DMA buffer.

If the destination address is located on OPB memory space, the DMA writes out the content of the DMA buffer with a series of single beat transfers if burst mode is disabled; or, it is written with a series of burst transfers if bursting is enabled.

Memory-to-peripheral transfers differ since the amount of data that will be requested by the peripheral is unknown. If the DMA buffer is disabled (DMA2P40_CRn[BEN] = 0) a discrete source memory read occurs for each element in the DMA transfer. Since this is inefficient, the buffer should only be disabled for low data rate transfers. When the 128-byte buffer is enabled the controller uses the setting in DMA2P40_CRn[PF] to prefetch 1, 2, 4, or 8 128-bit quadwords from the source memory. The DMA controller provides data from the buffer until the peripheral deasserts its request or the transfer completes. Whenever any of these conditions occurs, any unused data in the DMA buffer is discarded.

For both peripheral-to-memory and memory-to-peripheral transfers the DMA controller supports fixed length bursts of 2, 4, 8, or 16 data elements on the peripheral side of the transfer. When bursting is enabled (DMA2P40_CTCn[BTEN] = 1) an active DMA request (DMAReqn) causes the DMA controller to transfer the programmed number of data items (DMA2P40_CTCn[BSIZ]) in one atomic operation.

24.2.2 Memory-to-Memory Transfers

The DMA controller can perform either device-paced (hardware-initiated) or software-initiated memory-to-memory transfers. Device-paced memory transfers function identically to peripheral mode transfers, except that a chip select (PerCSn) serves as the DMA acknowledge instead of a DMAAckn output.

24.2.2.1 Device-Paced Memory Mode Transfers (Hardware Initiated)

Device-paced memory (DPM) is either devices attached to the EBC interface via the DMAReqn, External signals (including PerCSn and PerReady), or the internal DPM connected internally to the chip. Memory is any address accessible from the PLB or OPB, including PLB-mapped PCI Express address space, SDRAM memory, and memory connected to the external bus.

During a DPM mode transfer, the DPM device requests a DMA transfer by asserting a DMA request line. For internal DPMs, this signal is internal to the chip, while external peripherals use one of the DMAReqn lines. When the requesting channel has the highest priority of any active channel, the DMA function executes a DPM transfer, and the requesting device, if located in the external bus, receives an PerCSn signal. External DPMs use the PerReady signal to control the occurrence of the data Transfer.

There are two types of DPM transfers: DPM-to-memory and memory-to-DPM. A DPM-to-memory transfer reads data from a DMA DPM device, while a memory-to-DPM transfer writes data. In both cases, the DPM interface transfers data at the programmed width of the peripheral (DMA2P40_CRn[PW]). When the DMA buffer is disabled for the active channel (DMA2P40_CRn[BEN] = 0), each DPM transfer causes a corresponding memory operation. When buffering is enabled during a DPM-to-memory transfer, data is collected until the 128-byte buffer is full, the DPM deasserts DMAReqn, or the channel completes. The buffer contents are then written to the target memory as efficiently as possible.

If the Destination address is located on PLB space, the data is written to memory with one burst operation if the destination address is quadword aligned and the buffer is full; or, it is written with a combination of single beat and burst transfers, depending on the destination address alignment and the amount of data stored in the DMA buffer. If the destination address is located on OPB memory space, the DMA controller writes out the content of the DMA buffer with a series of single beat or burst transfers, depending on whether bursting is enabled.

Memory-to-DPM transfers differ since the amount of data that will be requested by the DPM is unknown. If the DMA buffer is disabled (DMA2P40_CRn[BEN] = 0), a discrete source memory read occurs for each element in the DMA transfer. Since this is inefficient, the buffer should only be disabled for low data rate transfers. When the 128-byte buffer is enabled, the controller uses the setting in DMA2P40_CRn[PF] to prefetch 1, 2, 4, or 8 128-bit quadwords from the source memory. The DMA controller provides data from the buffer until the DPM deasserts its request or the transfer completes. Whenever any of these conditions occurs, any unused data in the DMA buffer is discarded.

For both DPM-to-memory and memory-to-DPM transfers, the DMA controller supports fixed length bursts of 2, 4, 8, or 16 data elements on the peripheral side of the transfer. When bursting is enabled (DMA2P40_CTCn[BTEN] = 1), an active DMA request (DMAReqn) causes the DMA controller to transfer the programmed number of data items (DMA2P40_CTCn[BSIZ]) in one atomic operation.

24.2.2.2 Software-Initiated Memory-to-Memory Transfers

Software-initiated memory-to-memory transfers between memories with fixed timings provide the best overall performance. During a software-initiated transfer, the DMA controller knows the exact amount of data to be transferred. As a result, when the 128-byte DMA buffer is enabled (DMA2P40_CRn[BEN] = 1) the controller uses bursts as much as possible if the memory is located on PLB space. For memory located on OPB space, bursting has to be enabled (DMA2P40_CTCn[BTEN] = 1) for the DMA controller to burst. To ensure the highest bandwidth with memory locate on PLB space, source and destination addresses should be aligned on 128-byte boundaries.

There are two special cases of software-initiated memory-to-memory transfers.

- Source address increment bit is reset (DMA2P40_CRn[SAI] = 0).

  In this special case, the source memory device responds to a unique address. If the source memory device (FIFO-like device) is located on PLB space, bursting is not possible. If the source memory device is located on OPB space, bursting is possible if enabled (DMA2P40_CTCn[BTEN] = 1).

- Destination address increment bit is reset (DMA2P40_CRn[DAI] = 0).

  In this case, the destination memory device responds to a unique address. If the destination memory device (FIFO-like device) is located on PLB space, bursting is not possible. If the destination memory device is located on OPB space, bursting is possible if enabled (DMA2P40_CTCn[BTEN] = 1).

24.2.3 Scatter/Gather Transfers

Each of the four DMA channels supports scatter/gather transfers. This scatter/gather capability allows the chaining of multiple DMA controller operations within a channel. During a normal DMA operation software must program the control, count and control, source address, and destination address registers for each transfer. Scatter/gather transfers differ in that these registers are automatically loaded from a linked list data structure in system memory. When a channel completes one transfer, the DMA controller loads the next set of configuration values into the channel registers and the channel continues with the new programming.

24.3 DMA Configuration and Status Registers

Table 24-2 lists the DMA configuration and status registers, which are accessed using the **mtdcr** and **mfdcr** instructions. As example, the following PowerPC assembly code writes the control register for DMA channel 0 and then reads the DMA status register:

```
#define DMA2P40_CR0  0x100
#define DMA2P40_SR   0x120

    mtdcr     DMA2P40_CR0,r3            ! write r3 to channel 0 control register
    mfdcr     r4,DMA2P40_SR             ! read contents of status register into r4
```

The DMA configuration and status registers are readable at any time; however, since each register read requires a separate operation, it is not possible to guarantee that the values read from multiple registers correspond to a state that ever existed in the DMA controller. To illustrate, consider software that reads the destination address registers for channel 0 (DMA2P40_DAH0 and DMA2P40_DAL0) and the count for channel 0 (DMA2P40_CTC0). If the DMA controller updates the destination address or count between these operations, the values read differ from what is expected.

While reads can occur at any time, software must not write the configuration registers for any channel that is currently enabled (DMA2P40_CRn[CE] = 1). The only exception is that a channel may be disabled by reading the channel control register, clearing the channel enable bit, and then writing the new value to the control register. Note that before enabling the channel enable bit, the status has to be queried until the channel busy status bit is cleared. Once a channel is disabled, all of its configuration registers may be reprogrammed as desired.

Table 24-2. DMA Controller Configuration and Status Registers

Mnemonic	Name	DCR Number	Access	Page
DMA2P40_CR0	DMA Channel Control Register 0	0x200	R/W	829
DMA2P40_CTC0	DMA Count and Control 0	0x201	R/W	830
DMA2P40_SAH0	DMA Source Address High 0	0x202	R/W	832
DMA2P40_SAL0	DMA Source Address Low 0	0x203	R/W	832
DMA2P40_DAH0	DMA Destination Address High 0	0x204	R/W	832
DMA2P40_DAL0	DMA Destination Address Low 0	0x205	R/W	832
DMA2P40_SGH0	DMA Scatter/Gather Descriptor Address High 0	0x206	R/W	833
DMA2P40_SGL0	DMA Scatter/Gather Descriptor Address Low 0	0x207	R/W	833
DMA2P40_CR1	DMA Channel Control Register 1	0x208	R/W	829
DMA2P40_CTC1	DMA Count and Control 1	0x209	R/W	830
DMA2P40_SAH1	DMA Source Address High 1	0x20A	R/W	832
DMA2P40_SAL1	DMA Source Address Low 1	0x20B	R/W	832
DMA2P40_DAH1	DMA Destination Address High 1	0x20C	R/W	832
DMA2P40_DAL1	DMA Destination Address Low 1	0x20D	R/W	832
DMA2P40_SGH1	DMA Scatter/Gather Descriptor Address High 1	0x20E	R/W	833
DMA2P40_SGL1	DMA Scatter/Gather Descriptor Address Low 1	0x20F	R/W	833
DMA2P40_CR2	DMA Channel Control Register 2	0x210	R/W	829
DMA2P40_CTC2	DMA Count and Control 2	0x211	R/W	830
DMA2P40_SAH2	DMA Source Address High 2	0x212	R/W	832
DMA2P40_SAL2	DMA Source Address Low 2	0x213	R/W	832
DMA2P40_DAH2	DMA Destination Address High 2	0x214	R/W	832
DMA2P40_DAL2	DMA Destination Address Low 2	0x215	R/W	832
DMA2P40_SGH2	DMA Scatter/Gather Descriptor Address High 2	0x216	R/W	833
DMA2P40_SGL2	DMA Scatter/Gather Descriptor Address Low 2	0x217	R/W	833
DMA2P40_CR3	DMA Channel Control Register 3	0x218	R/W	829
DMA2P40_CTC3	DMA Count and Control 3	0x219	R/W	830
DMA2P40_SAH3	DMA Source Address High 3	0x21A	R/W	832
DMA2P40_SAL3	DMA Source Address Low 3	0x21B	R/W	832
DMA2P40_DAH3	DMA Destination Address High 3	0x21C	R/W	832
DMA2P40_DAL3	DMA Destination Address Low 3	0x21D	R/W	832
DMA2P40_SGH3	DMA Scatter/Gather Descriptor Address High 3	0x21E	R/W	833
DMA2P40_SGL3	DMA Scatter/Gather Descriptor Address Low 3	0x21F	R/W	833
DMA2P40_SR	DMA Status Register	0x220	R/Clear	833
DMA2P40_SGC	DMA Scatter/Gather Command	0x223	R/W	834
DMA2P40_SLP	DMA Sleep Mode	0x225	R/W	835
DMA2P40_POL	DMA Polarity Configuration	0x226	R/W	835

24.3.1 DMA Channel Control Registers (DMA2P40_CR0–DMA2P40_CR3)

DMA channel control registers (DMA2P40_CR0–DMA2P40_CR3) are used to configure and enable their respective DMA channels. Before a DMA channel can transfer data, the channel control, count and control, source address and destination address registers must be programmed. If a DMA channel is set up for scatter/gather transfers (DMA2P40_SGC[SSGn] = 1), the DMA channel control register is automatically loaded from memory (see *Scatter/Gather Transfers* on page 839).

Figure 24-3. DMA Channel Control Registers (DMA2P40_CR0–DMA2P40_CR3)

0	CE	Channel Enable 0 Channel is disabled 1 Channel is enabled	This field is automatically cleared when the transfer completes or an error occurs.
1	CIE	Channel Interrupt Enable 0 Disable interrupts from this channel 1 Enable interrupts from this channel	When enabled, interrupts can be generated for terminal count, end of transfer, and errors conditions. Each of these interrupt types must be individually enabled in DMA2P40_CTn.
2	TD	In peripheral mode: 0 Transfers are from memory-to-peripheral 1 Transfers are from peripheral-to-memory In device-paced memory-to-memory mode: 0 Peripheral is at the destination address 1 Peripheral is at the source address	TD is not used (don't care) for software-initiated memory-to-memory transfers.
3	PL	Peripheral Location/Destination Memory Location For memory-to-memory transfers: 0 Destination address located in PLB memory space 1 Destination address located in OPB memory space For peripheral/DPM-to-memory or memory-to-DPM/peripheral transfers: 0 Device located on external bus 1 Device located on the OPB.	
4:6	PW	Peripheral Width/Transfer Width 000 Byte (8 bits) 001 Halfword (16 bits) 010 Word (32 bits) 011 Doubleword (64 bits) 100 Quadword (128 bits)	Transfers involving peripheral, device-paced-memory, and OPB FIFO devices can only be byte, halfword, or word. This limitation also applies to transfers involving EBC memory when SA I = 0 or DAI = 0. PLB FIFO devices can only be quadword size.
7	DAI	Destination Address Increment 0 Do not increment destination address 1 After each data transfer increment the destination address by: • 1, for byte (8-bit) transfers • 2, for halfword (16-bit) transfers • 4, for word (32-bit) transfers • 8, for doubleword (64-bit) transfers • 16, for quadword (128-bit) transfers	Valid only when DEC = 0. Is required that DAI = 1 for peripheral-to-memory and device-paced memory-to-memory transfers since peripheral-to-FIFO type devices or DPM-to-FIFO type devices are not supported.
8	SAI	Source Address Increment 0 Do not increment source address 1 After each data transfer increment the source address by: • 1, for byte (8-bit) transfers • 2, for halfword (16-bit) transfers • 4, for word (32-bit) transfers • 8, for doubleword (64-bit) transfers • 16, for quadword (128-bit) transfers	Valid only when DEC = 0. Is required that SAI = 1 for memory-to-peripheral and memory-to-device-paced memory transfers since FIFO-to-peripheral/DPM transfers are not supported.

9	BEN	Buffer Enable 0 Disable DMA 128-byte buffer 1 Enable DMA 128-byte buffer	If BEN = 0 discrete read and write operations occur for each data transfer.
10:11	TM	Transfer mode 00 Peripheral 01 Reserved 10 Software-initiated memory-to-memory 11 Device-paced memory-to-memory	
12:13	PSC	Peripheral Setup Cycles 0–3	Number of PerClk cycles that the peripheral bus is idle from the last peripheral bus transaction to DMAAckn becoming active. Used only for the peripheral side of peripheral mode transfers.
14:19	PWC	Peripheral Wait Cycles 0–63	DMAAckn remains active for PWC+1 PerClk cycles. Used only for the peripheral side of peripheral mode transfers.
20:22	PHC	Peripheral Hold Cycles 0–7	The number of PerClk cycles between the time that DMAAckn becomes inactive until the peripheral bus is available for the next bus access. Used only during the peripheral side of peripheral mode transfers.
23	ETD	End-of-Transfer/Terminal Count (DMAEOTn) Pin Direction 0 DMAEOTn is an EOT input 1 DMAEOTn is a TC output	ETD must be set to 1 if the channel is configured for software-initiated memory-to-memory transfers.
24	TCE	Terminal Count (TC) Enable 0 Channel does not stop when TC is reached 1 Channel stops when TC is reached	If TCE = 1, it is required that ETD = 1.
25:26	CP	Channel Priority 00 Low priority 01 Medium low priority 10 Medium high priority 11 High priority	Actively requesting channels of the same priority are ranked in order by channel number, channel 0 having the highest priority. For more information, see "DMA Arbitration Transfer Priorities."
27:28	PF	Memory Read Prefetch Transfer 00 Prefetch 1 quadword (128-bits) 01 Prefetch 2 quadwords 10 Prefetch 4 quadwords 11 Prefetch 8 quadwords	Used only during memory-to-peripheral and memory to device paced memory transfers. To enable prefetching it is required that BEN = 1.
29	DEC	Address Decrement 0 SAI and DAI fields control memory address incrementing. 1 After each data transfer the memory address is decremented by the transfer width.	If DEC = 1, it is required that BEN = 0. This field is valid only for peripheral mode transfers (TM = 00).
30	SL	Source Address Location/Memory Location 0 PLB 1 OPB	For memory-to-memory transfers (TM = 1x) SL indicates whether the source memory controller is on the PLB or OPB. For peripheral mode transfers, this field denotes the location of the memory controller.
31		Reserved	

24.3.2 DMA Count and Control Registers (DMA2P40_CT0–DMA2P40_CT3)

The DMA count and control registers (DMA2P40_CT0–DMA2P40_CT3) contain the number of transfers remaining in the DMA transaction for their respective channels when DMAEOTn is programmed as a terminal count output, along with additional channel control information. In addition to the transfer count, the count and control registers contain fields used to enable interrupts for terminal count, end of transfer and error conditions. Other fields enable parity checking during the peripheral portion of peripheral type (DMA2P40_CRn[TM] = 00) transfers, define sub channel ID information, and enable bursting during peripheral and device-paced memory transfers. For additional details on bursting, see *Peripheral and Device Paced Memory Bursts* on page 837.

When a DMA channel is setup for scatter/gather transfers (DMA2P40_SGC[SSGn] = 1) the count and control register is automatically loaded from memory. For additional details see *Scatter/Gather Transfers* on page 839.

The value in the Transfer Count (TC) field of DMA count and control register is interpreted as the number of transfers of the width specified in DMA2P40_CRn[PW], not the total number of bytes. The maximum number of transfers is 1024K, and each transfer can be either 1, 2, 4, 8, or 16 bytes as programmed in DMA2P40_CRn[PW]. The maximum count of 1024K transfers is programmed by writing zero to DMA2P40_CTCn[TC].

The sub channel ID bits can be used to allow one DMA channel to support up to eight different peripherals. Bits 5:7 control the external logic that selects the current peripheral.

When DMAEOTn is programmed as an end-of-transfer input (DMA2P40_CRn[ETD] = 0), the channel will not stop when the count reaches zero. Instead, DMA2P40_CTCn[TC] continues to count down past zero until DMAEOTn is asserted. For DMA2P40_CRn[ETD] = 0, DMA2P40_CTCn[TCE] must be set to zero.

Figure 24-4 describes the DMA2P40_CT0–DMA2P40_CT3 register bits.

Figure 24-4. DMA Count and Control Registers (DMA2P40_CT0–DMA2P40_CT3)

0:1		Reserved	
2	TCIE	Terminal Count Interrupt Enable 0 Disable terminal count interrupts 1 Enable terminal count interrupts	Terminal Count interrupts are further qualified by DMA2P40_CR0-DMA2P40_CR3[CIE], UIC0_ER[12], UIC0_ER[13], UIC0_ER[14], and UIC0_ER[15].
3	ETIE	EOT Interrupt Enable 0 Disable end of transfer interrupts 1 Enable end of transfer interrupts	EOT interrupts are further qualified by DMA2P40_CR0-DMA2P40_CR3[CIE], UIC0_ER[12], UIC0_ER[13], and UIC0_ER[14], and UIC0_ER[15].
4	EIE	Error Interrupt Enable 0 Disable error interrupts 1 Enable error interrupts	Error interrupts are further qualified by DMA2P40_CR0-DMA2P40_CR3[CIE], UIC0_ER[12], UIC0_ER[13], and UIC0_ER[14], and UIC0_ER[15].
5:7	SID	Subchannel ID	See *Table 24-3*.
8	BTEN	Burst Enable 0 Disable bursting 1 Enable bursting	Controls bursting to and from peripheral devices and EBC-attached device-paced memory. See "Peripheral and Deviced-Paced Memory Bursts."
9:10	BSIZ	Burst Size 00 Burst of 2 01 Burst of 4 10 Burst of 8 11 Burst of 16	Determines the burst length used when BEN = 1. When BEN = 1, TC must be programmed to a multiple of BRSTSIZ. For OPB memory burst transfers, these bits determine the maximum burst size.
11	PCE	Parity Check Enable 0 Disable parity checking 1 Enable parity checking	Enables parity for peripheral mode transfers. See "Data Parity During DMA Peripheral Transfers."
12:31	TC	Transfer Count 0 – 1024K – 1	Programming TC = 0 results in the maximum count of 1024K transfers.

Table 24-3. DMA to PLB4 Channel Assignments

Channel	Subchannel	Internal Device Paced DMA Source/Destination
Channel 0	0	UART 0 Receive
	1	UART 2 Receive
Channel 1	0	UART 0 Transmit
	1	UART 2 Transmit
Channel 2	0	UART 1 Receive
	1	UART 3 Receive
Channel 3	0	UART 1 Transmit
	1	UART 3 Transmit

24.3.3 DMA Source Address Registers (DMA2P40_SAH/L0–DMA2P40_SAH/L3)

The DMA Source Address Registers (DMA2P40_SAHn and DMA2P40_SALn) contain the source address for memory-to-memory and memory-to-peripheral transfers. If a DMA channel is setup for scatter/gather transfers (DMA2P40_SGC[SSGn] = 1) the source address registers are automatically loaded from memory. For additional details *Scatter/Gather Transfers* on page 839

The source address must be aligned at the transfer width programmed in DMA2P40_CRn[TW], otherwise the error bit (DMA2P40_SR[RIn]) is set for the channel and no transfer occurs. If the source address increment bit in the channel's control register is set (DMA2P40_CRn[SAI]) the address is incremented by the transfer width after each data transfer. In contrast, if the channel is performing a memory-to-peripheral transfer and the address decrement bit is set (DMA2P40_CRn[DEC] = 1), the address is decremented by the transfer width after each transfer.

Figure 24-5. DMA Source Address High Registers (DMA2P40_SAH0–DMA2P40_SAH3)

0:31	USA	Upper 32-bits of source address for memory-to-memory and memory-to-peripheral transfers.	

Figure 24-6. DMA Source Address Low Registers (DMA2P40_SAL0–DMA2P40_SAL3)

0:31	LSA	Lower 32-bits of source address for memory-to-memory and memory-to-peripheral transfers.	

24.3.4 DMA Destination Address Registers (DMA2P40_DAH/L0–DMA2P40_DAH/L3)

DMA destination address registers (DMA2P40_DAH0–DMA2P40_DAH3 and DMA2P40_DAL0–DMA2P40_DAL3) contain the destination address for memory-to-memory and peripheral-to-memory transfers. When a DMA channel is configured for scatter/gather transfers (DMA2P40_SGC[SSGn] = 1) the destination address registers are automatically loaded from memory. For additional details see *Scatter/Gather Transfers* on page 839

The destination address must be aligned at the transfer width programmed in DMA2P40_CRn[TW], otherwise the error bit (DMA2P40_SR[RIn]) is set for the channel and no transfer occurs. If the destination address increment bit in the channel's control register is set (DMA2P40_CRn[DAI]) the address is incremented by the transfer width after

each data transfer. However, if the channel is performing a peripheral-to-memory transfer and the address decrement bit is set (DMA2P40_CRn[DEC] = 1), the destination address is decremented by the transfer width after each transfer.

Figure 24-7. DMA Destination Address High Registers (DMA2P40_DAH0–DMA2P40_DAH3)			
0:31	UDA	Upper 32-bits of destination address for memory-to-memory and peripheral-to-memory transfers.	

Figure 24-8. DMA Destination Address Low Registers (DMA2P40_DAL0-DMA2P40_DAL3)			
0:31	LDA	Lower 32-bits of destination address for memory-to-memory and peripheral-to-memory transfers.	

24.3.5 DMA Scatter/Gather Descriptor Address Regs (DMA2P40_SGH/L0–DMA2P40_SSGH/L3)

When a DMA channel is setup for scatter/gather transfers (DMA2P40_SGC[SSGn] = 1), the Scatter/Gather Descriptor Address Registers (DMA2P40_SGH0–DMA2P40_SGH3 and DMA2P40_SGL0–DMA2P40_SGL3) contain the memory address of the next scatter/gather descriptor table. Prior to starting a scatter/gather transfer, software must write the address of the channel's descriptor table to DMA2P40_SGHn and DMA2P40_SGLn. Once the scatter/gather transfer starts, DMA2P40_SGHn and DMA2P40_SGLn are automatically updated from the descriptor table. For additional details see *Scatter/Gather Transfers* on page 839.

Figure 24-9. DMA Scatter/Gather Desc Addr High Registers (DMA2P40_SGH0–DMA2P40_SGH3)			
0:31	USGD	Upper 32-bits of address of next scatter/gather descriptor table.	

Figure 24-10. DMA Scatter/Gather Desc Addr Low Registers (DMA2P40_SGL0–DMA2P40_SGL3)			
0:31	LSGD	Lower 32-bits of address of next scatter/gather descriptor table.	

24.3.6 DMA Status Register (DMA2P40_SR)

DMA Status Register (DMA2P40_SR) provides status information for each of the DMA channels. Bits in DMA2P40_SR are set in hardware, and can be either read or cleared by software. Clearing is performed by writing a word to DMA2P40_SR containing a 1 in any bit position to be cleared and 0 in all other bit positions.

The terminal count status (DMA2P40_SR[CSn]), end of transfer status (DMA2P40_SR[TSn]) and error status (DMA2P40_SR[RIn]) must be cleared for a DMA channel to operate. If a scatter/gather operation generates an interrupt for any of the above conditions, the channel pauses until software clears the associated status field(s) in DMA2P40_SR.

Figure 24-11. DMA Status Register (DMA2P40_SR)			
0:3	CS[0:3]	Channel 0-3 Terminal Count Status 0 Terminal count has not occurred 1 Terminal count has been reached	Set when the transfer count reaches 0 and the DMA2P40_CRn(TCE) bit is set.
4:7	TS[0:3]	Channel 0-3 End of Transfer Status 0 End of transfer has not been requested 1 End of transfer has been requested	Only valid for channels with DMA2P40_CRn[ETD] = 0 (no memory-to-memory transfers).
8:11	RI[0:3]	Channel 0-3 Error Status 0 No error 1 Error occurred	See Errors section for more information.
12:15	IR[0:3]	Internal DMA Request 0 No internal DMA request pending 1 Internal DMA request is pending	
16:19	ER[0:3]	External DMA Request 0 No external DMA request pending 1 External DMA request is pending	
20:23	CB[0:3]	Channel Busy 0 Channel is idle 1 Channel currently active	During scatter/gather fetches, these bits are active.
24:27	SG[0:3]	Scatter/Gather Status 0 No scatter/gather operation in progress 1 Scatter/gather operation in progress	For multiple scatter/gather links, the scatter/gather status bit is set when the first link is loaded and is kept active until the last link is completed.
28:31		Reserved	

24.3.7 DMA Scatter/Gather Command Register (DMA2P40_SGC)

The DMA Scatter/Gather Command Register (DMA2P40_SGC) is a 32-bit register, of which 12-bits are implemented. Bits 0:3 are the Start Scatter/Gather Enable bits for channels 0 to 3, bits 4:7 determine whether a given channel's Scatter/Gather Descriptor Table resides in PLB or OPB address space, and bits 16:19 are the corresponding Enable Mask bits for the Start Scatter/Gather Enable bits. Setting a Start Scatter/Gather Enable bit causes the selected channel to begin a scatter/gather operation, while writing a 0 stops the Scatter/Gather operation. To start or stop a specific Scatter/Gather channel, the corresponding Enable Mask bit must be set to 1; otherwise, the register holds the previous value. Note that halting a scatter/gather transfer prevents the channel from continuing to the next descriptor, but does not stop the transfer currently in progress.

Upon completion of a scatter/gather sequence of transfers the DMA controller clears DMA2P40_SGC[SSGn]. If an error occurs when the DMA controller is reading the Scatter/Gather descriptor table, DMA2P40_SGC[SSGn] is cleared for the affected channel, and the channel's error status bit (DMA2P40_SR[RIn]) is set. For additional details see *Scatter/Gather Transfers* on page 839.

Figure 24-12. DMA Scatter/Gather Command Register (DMA2P40_SGC)			
0:3	SSG[0:3]	Start Scatter/Gather for channels 0-3 0 Scatter/gather support is disabled 1 Scatter/gather support is enabled	To start a scatter/gather operation for channel n, EM[n] must also be set.
4:7	SGL[0:3]	Scatter/Gather Descriptor Table Location for channels 0-3 0 PLB memory space 1 OPB memory space	A channel's descriptor table may not cross between PLB and OPB address space. See *Scatter/Gather Transfers* on page 839.
8:15		Reserved	
16:19	EM[0:3]	Enable Mask for channels 0-3 0 Writes to SSG[n] and SGL[n] are ignored 1 Allow writing to SSG[n] and SGL[n]	To write SSG[n] or SGL[n], EM[n] must be set. Otherwise, writing SSG[n] or SGL[n] has no effect.
20:31		Reserved	

24.3.8 DMA Sleep Mode Register (DMA2P40_SLP)

The Sleep Mode Register (DMA2P40_SLP) enables the DMA controller to enter sleep (low-power) mode and programs the number of PLB clock cycles to wait when the controller is idle before going to sleep. The DMA controller only goes to sleep when no DMA channels are enabled, no scatter/gather operation is pending, and no configuration or status (DCR) register operations are in progress. Reading or writing any of the DMA DCRs awakens the DMA controller.

To enable sleep mode set DMA2P40_SLP[SME] and CPM0_ER[DMA]. When sleep mode is enabled and the DMA controller becomes idle the 10-bit idle timer begins counting down from the programmed value. Only the upper five bits of the idle counter are programmable; the lower five bits are hard coded to 0b11111; therefore, the minimum granularity of the idle timer is 32 PLB clock cycles. When the counter reaches zero, the controller is placed in sleep mode.

Figure 24-13. DMA Sleep Mode Register (DMA2P40_SLP)			
0:4	IDU	Idle Timer Upper 0-31	Upper 5 bits of the idle timer.
5:9	IDL	Idle Timer Lower Hard coded to 0b11111	Lower 5-bit portion of the idle timer. Writing this field has no effect.
10	SME	Sleep Mode Enable 0 Sleep disabled 1 Sleep enabled	If SME = 1, also set CPM0_ER[DMA] to enable the Clock and Power Management macro to put the DMA controller to sleep.
11:31		Reserved	

24.3.9 DMA Polarity Configuration Register (DMA2P40_POL)

The Polarity Configuration Register (DMA2P40_POL) is used to set the polarity (active state) of the external DMA I/O signals: DMAReqn, DMAAckn, and DMAEOTn. As shown in *Figure 24-14*, if a bit in DMA2P40_POL is zero, the corresponding signal is active high, otherwise the signal is active low.

Whenever any of the EOT polarities are changed (DMA2P40_POL[EnP), software must subsequently clear the corresponding EOT status bits in the DMA Status Register (DMA2P40_SR[TS0:3]) prior to enabling the associated DMA channel. This is necessary to prevent a channel from being disabled because of an incorrect EOT status stored in the status bits.

Figure 24-14. DMA Polarity Configuration Register (DMA2P40_POL)			
0	R0P	DMAReq0 Polarity 0 Active high 1 Active low	
1	A0P	DMAAck0 Polarity 0 Active high 1 Active low	
2	E0P	DMAEOT0 Polarity 0 Active high 1 Active low	
3	R1P	DMAReq1 Polarity 0 Active high 1 Active low	
4	A1P	DMAAck1 Polarity 0 Active high 1 Active low	
5	E1P	DMAEOT1 Polarity 0 Active high 1 Active low	
6	R2P	DMAReq2 Polarity 0 Active high 1 Active low	
7	A2P	DMAAck2 Polarity 0 Active high 1 Active low	
8	E2P	DMAEOT2 Polarity 0 Active high 1 Active low	
9	R3P	DMAReq3 Polarity 0 Active high 1 Active low	
10	A3P	DMAAck3 Polarity 0 Active high 1 Active low	
11	E3P	DMAEOT3 Polarity 0 Active high 1 Active low	
12:31		Reserved	

24.4 Channel Priorities

The priority of DMA transfers is controlled on a per-channel basis by the channel priority field in the channel control register. *Table 24-4* shows the different priority settings for DMA2P40_CRn[PW].

Table 24-4. DMA Transfer Priorities

DMA2P40_CRx[CP]	Priority Level
0b00	Low
0b01	Medium Low
0b10	Medium High
0b11	High

These priorities serve two purposes. First, the DMA controller arbitrates among all actively requesting channels and selects the highest priority one for service. If multiple channels request at the same priority, the arbiter selects the lowest numbered channel for service. Secondly, DMA2P40_CRn[CP] determines the priority of the internal PLB transactions that the DMA controller uses to read and write data. During a Scatter/Gather transfer, the next descriptor is read using the priority current programmed in DMA2P40_CRn[CP].

24.5 Data Parity During DMA Peripheral Transfers

The DMA controller works in conjunction with the peripheral bus controller (EBC) to generate and check parity during peripheral mode DMA transfers. When DMA2P40_CTCn[PCE] = 1 parity checking is enabled for peripheral mode transfers on channel n.

During memory-to-memory transfers any data error checking and/or correction is dependent on the configuration of the relevant memory controller. For example, if ECC is enabled on the SDRAM controller, only uncorrectable errors are reported to the DMA controller. Similarly, an EBC memory bank with parity enabled will report an error if a parity error is encountered during a memory read operation.

24.6 Peripheral and Device Paced Memory Bursts

Normally, peripheral and device paced (hardware initiated) DMA transfers move one data item at a time. The device requesting DMA service asserts DMAReqn and the DMA controller returns DMAAckn for a peripheral transfer or PerCSn for a device-paced transfer. This process continues for each item of the DMA transfer.

To improve performance, the DMA controller supports fixed-length bursts during the peripheral portion of peripheral mode DMA transfers and on the device-paced side of device-paced memory-to-memory transfers. When bursting is enabled (DMA2P40_CTCn[BTEN] = 1), the DMA controller responds to an active DMAReqn by reading or writing DMA2P40_CTCn[BSIZ] data items in one atomic, non-interruptible operation. The timing on the external interface is dictated by DMA2P40_CRn for peripheral mode transfers and EBCx_BnAP for device-paced memory-to-memory transfers. The external device is required to keep DMAReqn asserted until at least the first transfer of the burst. During the final transfer of the fixed-length burst DMAReqn is sampled on the PerClk edge where DMAReqn or PerCSn goes inactive. To prevent another burst from starting, DMAReqn must be sampled inactive at this point. This requirement means that single-cycle DMA transfers require that DMAReqn be deasserted via combinational logic.

When a DMA channel has bursting enabled (DMA2P40_CTCn[BTEN] = 1) it is required that the prefetch count (DMA2P40_CRn[PF]) be sufficient to complete at least one fixed-length burst transfer. In addition, the transfer count (DMA2P40_CTCn[TC]) must be a multiple of the burst size (DMA2P40_CTCn[BSIZ]). If either of these requirements is not met, an error results, no data transfer occurs, and the channel is disabled. For additional details see *Errors* on page 838

24.7 Errors

The DMA controller detects and reports five types of errors: address alignment, burst count, burst prefetch, PLB/OPB time-out, and slave errors. The DMA controller reports errors through the channel error status bit in the DMA status register (DMA2P40_SR[RIn]). Whenever the error status bit for a channel is set, the channel enable bit (DMA2P40_CRn[CE]) is cleared, disabling the channel. An interrupt signal is also presented to the interrupt controller if DMA2P40_CRn[CIE] and DMA2P40_CTCn[EIE] are set. For more information on interrupt processing, see *DMA Interrupts* on page 839

When an error is detected during the execution of an external cycle, the DMA terminates the transfer and deasserts all external signals.

When the DMA controller has multiple channels active, an error may be reported on the current channel which was in actuality caused by a previously active channel. This causes the current channel to have its error status bit set; therefore, for deterministic error analysis with multiple DMA channels active, the slave bus controller's error status registers (the bus error address register in particular) must be queried to isolate the actual channel that encountered the error. In any case, the channel causing the errors will eventually cause all active channels, including itself, to be disabled.

24.7.1 Address Alignment Error

The source address (DMA2P40_SAHn || DMA2P40_SALn) and destination address (DMA2P40_DAHn || DMA2P40_DALn) registers must be aligned to the programmed transfer width (DMA2P40_CRn[PW]).The address alignment rules are outlined in *Table 24-5*. In addition, when a channel is configured for scatter/gather transfers, the scatter/gather table must be quadword-aligned. If the source, destination, and scatter/gather address registers are not appropriately aligned, an error occurs immediately after the channel is enabled.

Table 24-5. Address Alignment Requirements

DMA2P40_CRx[PW] Setting	Required Alignment for: Source Address (DMA2P40_SAHn || DMA2P40_SALn) Destination Address (DMA2P40_DAHn || DMA2P40_DALn)
0b000	Byte (8-bit)
0b001	Halfword (16-bit)
0b010	Word (32-bit)
0b011	Doubleword (64-bit)
0b100	Quadword (128-bit)

24.7.2 Burst Count Error

If peripheral/device-paced bursting is enabled for a particular DMA channel (DMA2P40_CTCn[BTEN] = 1), the programmed transfer count (DMA2P40_CTCn[TC]) must be a multiple of the burst size (DMA2P40_CTCn[BSIZ]).

24.7.3 Burst Prefetch Error

If peripheral/device-paced bursting is enabled for a particular DMA channel (DMA2P40_CTCn[BTEN] = 1), the programmed prefetch count (DMA2P40_CRn[PF]) must be sufficient to perform at least one fixed-length burst. More precisely, DMA2P40_CRn[PF] must prefetch at least DMA2P40_CTCn[BSIZ] * DMA2P40_CRn[PW] bytes.

24.7.4 PLB or OPB Timeout

The DMA controller uses PLB and OPB operations to read and write memory. A PLB or OPB timeout results if the DMA controller attempts to access a non-existent memory location. This will occur if the source, destination or scatter/gather address registers do not map to valid memory locations.

24.7.5 Slave Transfer Errors

If the DMA controller detects an error from a slave, it stops the execution of the current transfer, disables the channel, and then reports an error. If a signal is asserted by the PLB slave as a result of such error, the DMA finishes any active read/write pair transfer before disabling the channel. An SDRAM uncorrectable ECC error and an EBC bank protection error are examples of slave errors.

24.8 DMA Interrupts

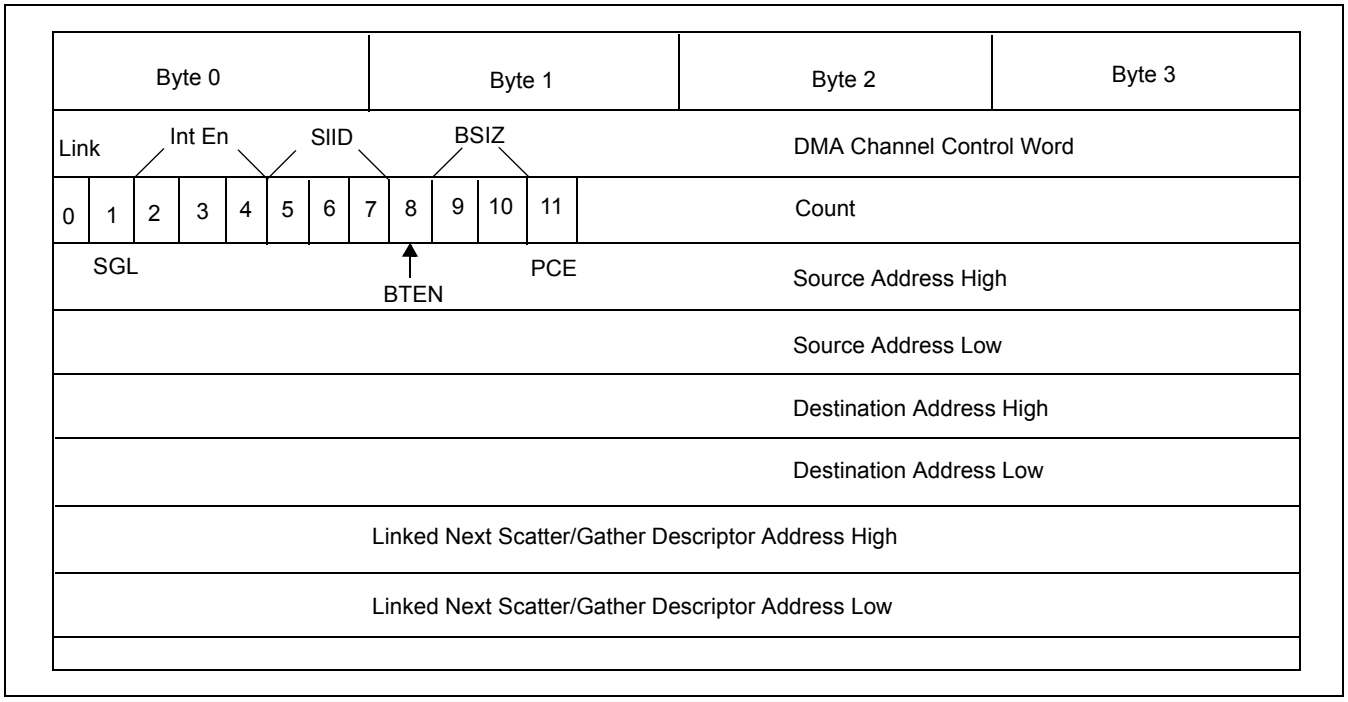
Each DMA channel can generate interrupts for end of transfer, terminal count and error conditions. All interrupts from a particular DMA channel are enabled by setting the channel enable bit in the channel's control register (DMA2P40_CRn[CIE] = 1) and enabling the specific type of interrupt in the count and control register: end of transfer (DMA2P40_CTCn[ETIE]), terminal count (DMA2P40_CTCn[TCIE]), and error (DMA2P40_CTCn[EIE]). When an interrupt occurs for a given channel, the DMA controller sends a signal to the Universal Interrupt Controller. For the CPU to take an exception, interrupts from the particular DMA channel must be enabled in the interrupt controller's interrupt enable register (UIC0_ER[DMA2P40] = 1). Also, the CPU machine state register's interrupt enable bit must be enabled for the appropriate interrupt type (critical or non-critical), MSR[EE,CE].

When the DMA controller generates an interrupt, the interrupt remains active until the appropriate bits are cleared in the DMA Status Register (DMA2P40_SR). In addition, interrupts from a channel performing a scatter/gather transfer cause the channel to pause until the interrupt is cleared.

24.9 Scatter/Gather Transfers

With a normal DMA transfer it is necessary to program a channel's control, count and control, source, and destination registers for each transfer. The scatter/gather capability of the DMA controller provides a more efficient solution for applications that require multiple transactions on a single DMA channel. Instead of individually programming a channel's registers, software creates a set (linked list) of descriptor tables in system memory. *Figure 24-15* illustrates the required table format.

Figure 24-15. Scatter/Gather Descriptor Table



The most significant bit of word 2, the transfer count and configuration register value, is the link (LK) bit. Bit number 1 (SGL) of the same word is the scatter/gather locator bit. It determines the location of the next scatter/gather table (0 = PLB space, 1 = OPB space).Other than these bits, the contents of the descriptor table are identical to the associated register definitions.

To configure a channel for a scatter/gather transfer the DMA Scatter/Gather Descriptor Address Registers (DMA2P40_SGHn and DMA2P40_SGLn) for the channel are set to the address of the first descriptor table, which must be quadword (16 byte) aligned. The linked list of descriptor tables for a given programming of the DMA controller must not cross between PLB or OPB address space. The location of the linked list is programmed into DMA2P40_SGC[SGLn].

To begin a scatter/gather transfer, software writes DMA2P40_SGC with the enable mask (DMA2P40_SGC[EMn]) set, the start scatter/gather bit (DMA2P40_SGC[SSGn]) set and indicates the location (PLB or OPB) of the descriptor tables via DMA2P40_SGC[SGLn]. The DMA controller then reads the descriptor table at address (DMA2P40_SGHn || DMA2P40_SGLn) and updates the DMA controller registers as shown in *Table 24-6*. Upon receiving the data from the scatter/gather descriptor table, the channel's terminal count status bit (DMA2P40_SR[TCn]) and end of transfer status bit (DMA2P40_SR[EOTn]) are automatically cleared.

Table 24-6. DMA Registers Loaded from Scatter/Gather Descriptor Table

Descriptor Table Entry	Register Loaded
Channel Control Word	DMA2P40_CRn
Source Address	DMA2P40_SAHn and DMA2P40_SALn
Destination Address	DMA2P40_DAHn and DMA2P40_DALn
Count and Control	DMA2P40_CTCn
Next Descriptor Address	DMA2P40_SGHn and DMA2P40_SGLn

After loading the channel's registers from the descriptor table, the transfer functions as a normal non-scatter/gather operation.

If the LK (link) bit was not set the scatter/gather process stops when the current transfer completes. Otherwise, the DMA controller reads the descriptor table at address (DMA2P40_SGHn || DMA2P40_SGLn) and the process repeats.

24.10 Programming the DMA Controller

Before the DMA controller can transfer data it must be configured, both globally and on a per-channel basis. Global settings include the DMA Polarity Register (DMA2P40_POL) and DMA Sleep Mode Register (DMA2P40_SLP). For most applications, these registers should be configured when the DMA controller is first initialized. To prevent spurious activity resulting from changing the active level for DMAReqn, DMAAckn, or DMAEOTn, a channel's configuration in the Polarity Register should not be altered when the channel is enabled (DMA2P40_CRn[CE] = 1).

Each channel has Control (DMA2P40_CRn), Count and Control (DMA2P40_CTCn), Source Address (DMA2P40_SAHn and DMA2P40_SALn), Destination Address (DMA2P40_DAHn and DMA2P40_DALn), and Scatter/Gather Descriptor Address (DMA2P40_SGHn and DMA2P40_SGLn) registers. The type of DMA transfer determines which of these registers must be programmed and what causes the channel to start. In all cases, the terminal count (CSn), end of transfer (TSn) and error status (RIn) bits in the DMA Status Register (DMA2P40_SR) must be cleared or the channel will not start.

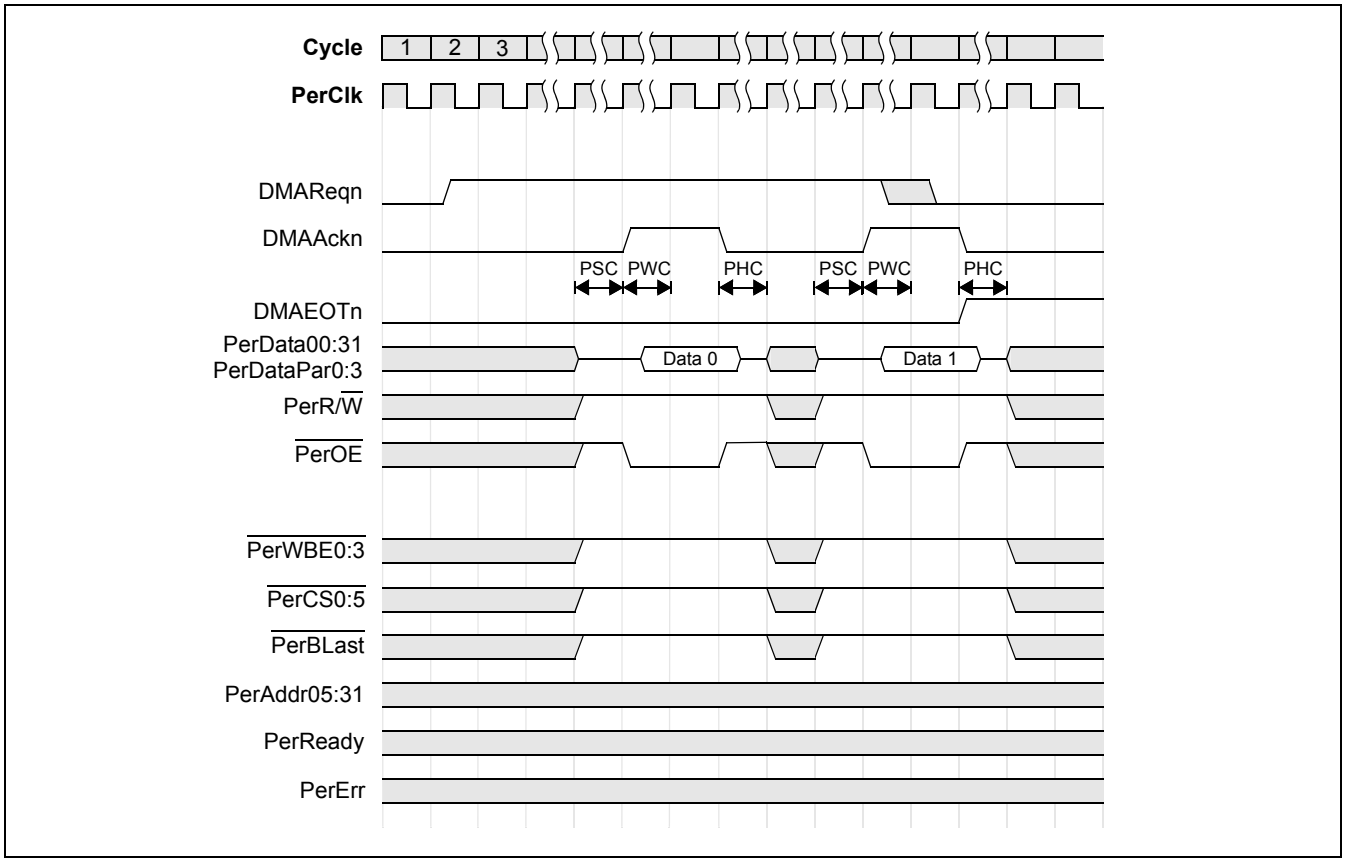
The programming information that follows assumes that the DMA controller is operating in non-scatter/gather mode. To use scatter/gather transfers the channel configuration data must be written into a set of descriptor tables in system memory. See *Scatter/Gather Transfers* on page 839 for additional details.

24.10.1 Peripheral Mode Transfers

DMA peripherals are either devices attached to the EBC interface via the DMAReqn and DMAAckn lines, or the internal serial ports (UART0:3). During a peripheral mode transfer, a peripheral asserts DMAReqn to request a DMA transfer. For metastability protection DMAReqn is double latched in the DMA upon assertion, and sampled with a single latch on the deassertion.

Timings on the memory-access portion of peripheral mode DMA transfers are governed by the configuration of the associated memory controller. In contrast, timing during the peripheral portion of the transfer is controlled by the PSC, PWC and PHC fields in the DMA Channel Control Registers. The effect of these parameters on peripheral timings is illustrated in *Figure 24-16* and *Figure 24-17*. Although shown as active high, the polarity (active state) of DMAReqn, DMAAckn, and DMAEOTn are programmableby means of the DMA Polarity Register (DMA2P40_POL).

Figure 24-16. Peripheral-to-Memory DMA



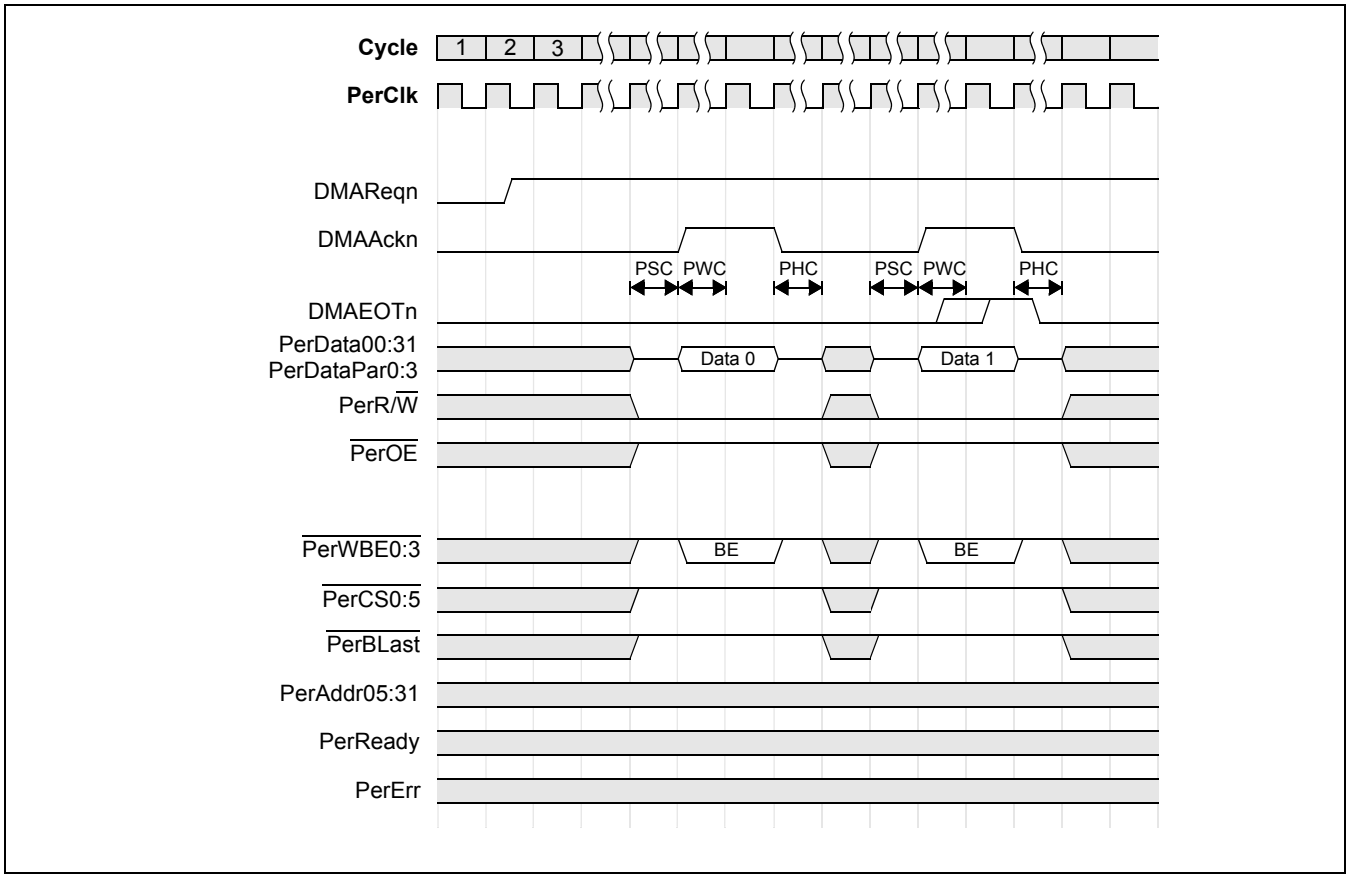
Peripheral setup cycles (DMA2P40_CRn[PSC]) provide a delay between any previous operation on the peripheral bus and DMAAckn becoming active. Following the setup time, DMAAckn is driven active for (DMA2P40_CRn[PWC] + 1) PerClk cycles. During peripheral-to-memory transfers, read data from the peripheral is sampled on the last cycle where DMAAckn is active. After DMAAckn becomes inactive the peripheral bus is held idle for DMA2P40_CRn[PHC] PerClk cycles.

The second transfer in *Figure 24-16* illustrates the required DMAReqn timing to prevent a subsequent DMA transfer. For all peripheral mode transfers, DMAReqn must be sampled inactive at the end of the last PerClk cycle where DMAAckn is active.

The EOTn/TCn I/O can be configured either as an end of transfer input (DMA2P40_CRn[ETD] = 0) or a terminal count output (DMA2P40_CRn[ETD] = 1). When programmed as a terminal count output, EOTn/TCn is asserted in the cycle after DMAAckn became inactive and the channel's count (DMA2P40_CTCn[TC]) reached zero. EOTn/TCn remains active until the terminal count status bit is cleared in the DMA status register (DMA2P40_SR[CSn]).

If EOTn/TCn is configured as an end of transfer input (DMA2P40_CRn[ETD] = 0), EOTn/TCn must be sampled active one external cycle before the last DMAAckn for external peripherals or during the last DMAAckn cycle for internal peripherals. If the channel is configured for scatter/gather transfers EOTn/TCn should be immediately deasserted to prevent the subsequent transfer from ending prematurely. *Figure 24-17* shows the required timing.

Figure 24-17. Memory-to-Peripheral DMA Transfer



For both peripheral-to-memory and memory-to-peripheral transfers the transfer width (DMA2P40_CRn[PW]) must be set to the data bus width of the peripheral. This is because the DMA controller does not pack or unpack data on the peripheral side of transaction.

24.10.1.1 Peripheral-to-Memory Transfer

To perform a peripheral-to-memory DMA transfer from an EBC-attached DMA peripheral:

1. Set destination address register (DMA2P40_DAHn and DMA2P40_DALn) to the desired memory location. The address must be aligned to the programmed transfer width (DMA2P40_CRn[PW]), otherwise an alignment error will occur.
2. In the count and control register:
    1. Program the desired count in the transfer count, TC, field.
    2. Optionally enable parity checking, PCE = 1.
    3. Optionally enable bursting, BRSTEN = 1, and set the burst size field, BRSTSIZ, as desired.
3. Clear the channel's status bits in the DMA status register (DMA2P40_SR).
4. In the channel control register (DMA2P40_CRn):
    1. Optionally enable the DMA buffer, BEN = 1.
    2. Set the destination address increment, DAI = 1.

3. Set the transfer mode to peripheral, TM = 0b00.
4. Set the peripheral location to external, PL = 0.
5. Set the setup/wait/hold times for the transfer.
6. Set interrupts if required.
7. Set the transfer direction to peripheral-to-memory, TD = 1.
8. Enable the channel CE = 1.

Once the DMA channel is active, the peripheral initiates a transfer by activating the DMAReqn pin for the channel. The chip then activates the DMAAckn pin to read data from the peripheral. This continues until either a terminal count or end of transfer condition occurs.

24.10.1.2 Memory-to-Peripheral Transfer

To perform a memory-to-peripheral DMA transfer to an EBC-attached DMA peripheral:

1. Set source address register (DMA2P40_SAHn and DMA2P40_SALn) to the desired memory location. The address must be aligned to the programmed transfer width (DMA2P40_CRn[PW]), otherwise an alignment error will occur.
2. In the count and control register:
    1. Program the desired count in the transfer count, TC, field.
    2. Optionally enable parity checking, PCE = 1.
    3. Optionally enable bursting, BRSTEN = 1, and set the burst size field, BRSTSIZ, as desired.
3. Clear the channel's status bits in the DMA status register (DMA2P40_SR).
4. In the channel control register (DMA2P40_CRn):
    1. Optionally enable the DMA buffer, BEN = 1, and set the desired prefetch count, PF.
    2. Set the source address increment, SAI = 1.
    3. Set the transfer mode to peripheral, TM = 0b00.
    4. Set the peripheral location to external, PL = 0.
    5. Set the transfer direction to memory-to-peripheral TD = 0.
    6. Enable the channel, CE = 1.

Once the DMA channel is active, the peripheral initiates a transfer by activating the DMAReqn pin for the channel. The chip then reads the source memory and subsequently activates the DMAAckn pin to write data to the peripheral. This continues until either a terminal count or end of transfer condition occurs.

24.10.2 Memory-to-Memory Transfers

Memory-to-memory transfers can be initiated either by software or by an external device. If initiated via software, the transfer begins as soon as the channel is configured and enabled. When initiated by hardware (also known as a device-paced memory-to-memory transfer), software configures the channel for a memory-to-memory move and transfers begin when an external device places an active request on the channel request line, DMAReqn.

24.10.2.1 Hardware-Initiated (Device-Paced) Memory-to-Memory Transfers

To perform a device-paced memory-to-memory DMA transfer:

1. Set the transfer width (DMA2P40_CRn[PW]) to the width of the device-paced memory.
2. Set the source (DMA2P40_SAHn and DMA2P40_SALn) and destination (DMA2P40_DAHx and DMA2P40_DALn) address registers to the desired memory locations. These addresses must be aligned to the programmed transfer width (DMA2P40_CRn[PW]), otherwise an alignment error will occur.
3. In the count and control register:

Caution: Program the desired count in the transfer count, TC, field.

Caution: Optionally enable bursting, BRSTEN = 1, and set the burst size field, BRSTSIZ, as desired.

4. Clear the channel's status bits in the DMA status register (DMA2P40_SR).
5. In the channel control register (DMA2P40_CRn):
   a. Optionally enable the DMA buffer, BEN = 1, and set the desired prefetch count, PF.
   b. If the device-paced memory is at the source memory location set PL = 1.
   c. Set the source address increment, SAI, and destination address increment, DAI, as desired.
   d. Set the transfer mode to device-paced memory-to-memory, TM = 0b11.
   e. If the memory controller is on the OPB set the source location bit, SL = 1. If the controller is on the PLB, clear SL.
   f. Enable the channel, CE = 1.

Once the DMA channel is configured for this mode, the external device initiates a transfer by activating the DMAReqn input. The chip then reads the source memory, buffers the data in the DMA controller and then outputs the data to the destination memory address. Transfers continue as long as the controlling device maintains an active signal on DMAReqn and the channel count register (DMA2P40_CTCn) is non-zero. To pause a device paced memory-to-memory transfer, the controlling device must deassert DMAReqn one PerClk cycle before the last cycle in the device-paced memory access.

24.10.2.2 Software-Initiated Memory-to-Memory Transfers (Non-Device Paced)

To perform a software-initiated memory-to-memory DMA transfer:

1. Set the transfer width (DMA2P40_CRn[PW]) as desired.
2. Set the source (DMA2P40_SAHn and DMA2P40_SALn) and destination (DMA2P40_DAHn and DMA2P40_DALn) address registers to the desired memory locations. These addresses must be aligned to the programmed transfer width (DMA2P40_CRn[PW]), otherwise an alignment error will occur.
3. Program the transfer count field in the control and count register (DMA2P40_CTCn[TC]) for the number of transfers.
4. Clear the channel's status bits in the DMA status register (DMA2P40_SR).
5. In the channel control register (DMA2P40_CRn):
   a. Optionally enable the DMA buffer, BEN = 1.
   b. Set the source address increment, SAI, and destination address increment, DAI, as desired.
   c. Set the transfer mode to software-initiated memory-to-memory, TM = 0b10.
   d. If the source memory controller is on the OPB set the source location bit, SL = 1. If the controller is on the PLB, clear SL.

e. If the destination memory controller is on the OPB set the PL = 1. If the memory controller is on the PLB clear the PL bit.

f. Enable the channel, CE = 1.

Once the channel is enable the DMA controller transfers data from source to destination until the channel count reaches zero. Note that memory-to-memory transfers initiated by software do not use DMAReqn or DMAAckn.

25. External Bus Controller

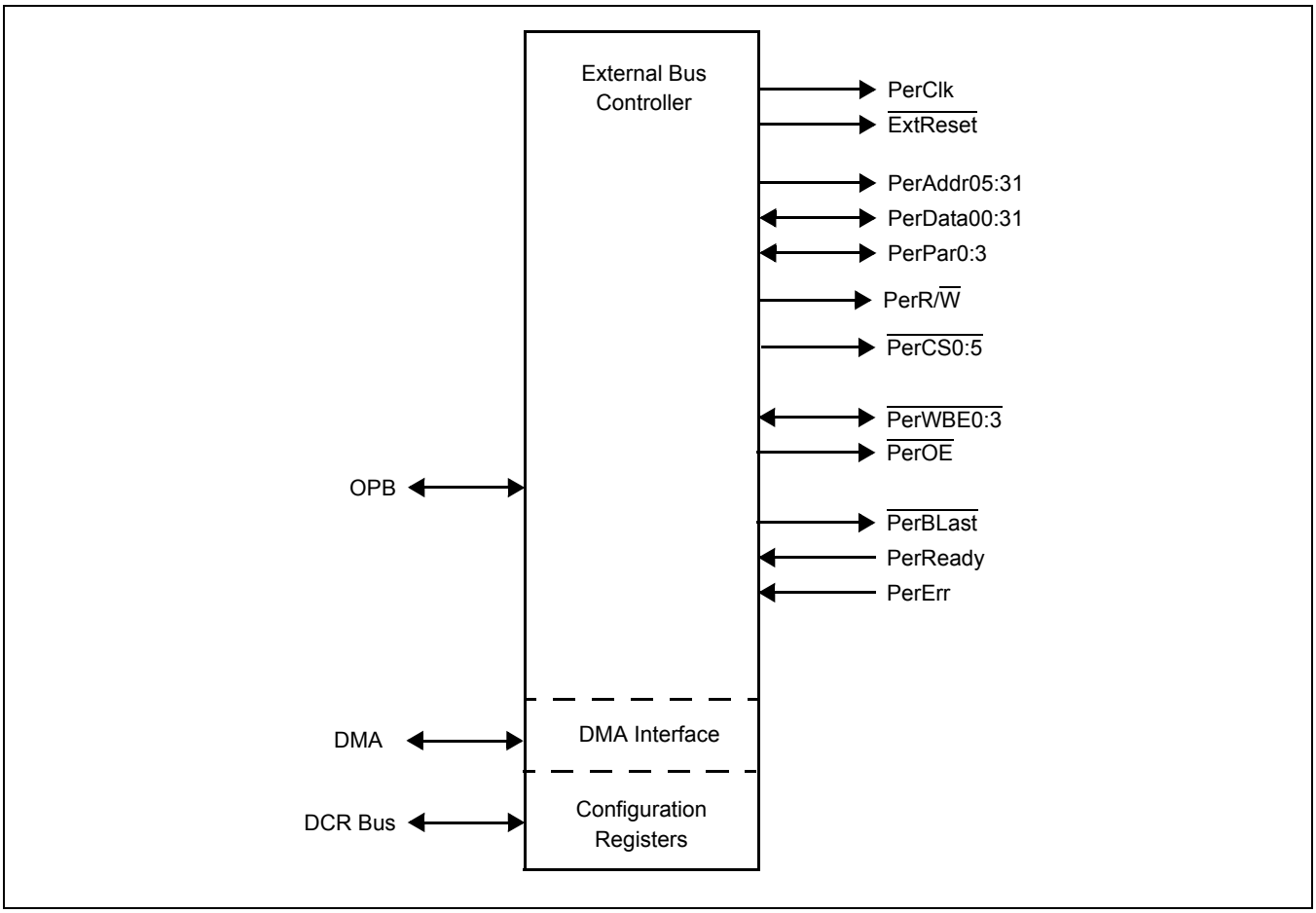
The PPC460EX/EXr/GT External Bus Controller (EBC) attached to the OPB provides direct attachment for most SRAM/Flash type memory and peripheral devices. The interface minimizes the amount of external glue logic needed to communicate with memory and peripheral devices. This reduces the embedded system device count, circuit board area, and cost.

To eliminate off-chip address decoding, the EBC provides four programmable chip selects that enable system designers to locate memory and peripherals within the PPC460EX/EXr/GT memory map. Chip select, data bus, and associated control signal timings are programmable for both single and burst transfers. For peripherals with variable timing requirements, the EBC supports device-paced transfers with optional bus-timeout. System design is further simplified through dynamic bus sizing, which supports seamlessly attaching 8-, 16-, and 32-bit wide memories and peripherals. Whenever a size mismatch exists between a read or write operation and the externally attached device, the EBC automatically packs or unpacks data as appropriate.

25.1 Interface Signals

Figure 25-1 illustrates the signal I/O between the EBC and the external peripheral bus.

Figure 25-1. External Bus Controller Signals



The usage along with the state of these signals during and after a reset is as follows:

Table 25-1. EBC Signal Usage and State During/Following a Chip or System Reset

Signal	ExtReset = 0	ExtReset = 1	Usage
PerClk	See Note	Toggling	Peripheral bus clock. During an EBC transfer, all EBC signal transitions and data sampling occurs synchronous to PerClk.
PerAddr05:31	High impedance	Last Address	Peripheral address bus. PerAddr05 is the most significant bit.
PerData00:31	High impedance	00000000	Peripheral data bus. PerData0 is the most significant bit.
PerPar0:3	High impedance	0000	Peripheral parity bus. The EBC implements odd parity.
$\overline{\text{PerCS0:5}}$	High impedance	0b111	Chip selects. Use CS0 to select the device when attaching boot device to the peripheral bus.
PerR/$\overline{\text{W}}$	High impedance	1	Read not write.
$\overline{\text{PerWBE0:3}}$	High impedance	0b111	Write byte enables or read/write byte enables.
$\overline{\text{PerOE}}$	High impedance	1	Output enable.
$\overline{\text{PerBLast}}$	High impedance	1	Burst Last. Active during non-burst operations and the last transfer of a burst access.
PerReady	Input	Input	An input to allow external peripherals to perform device-paced transfers.
PerErr	Input	Input	Peripheral data error input. Sampled during the data transfer only.
Note: PerClk is stable a minimum of 20 µs prior to the deassertion of $\overline{\text{ExtReset}}$, assuming SysClk is 66 MHz or lower.			

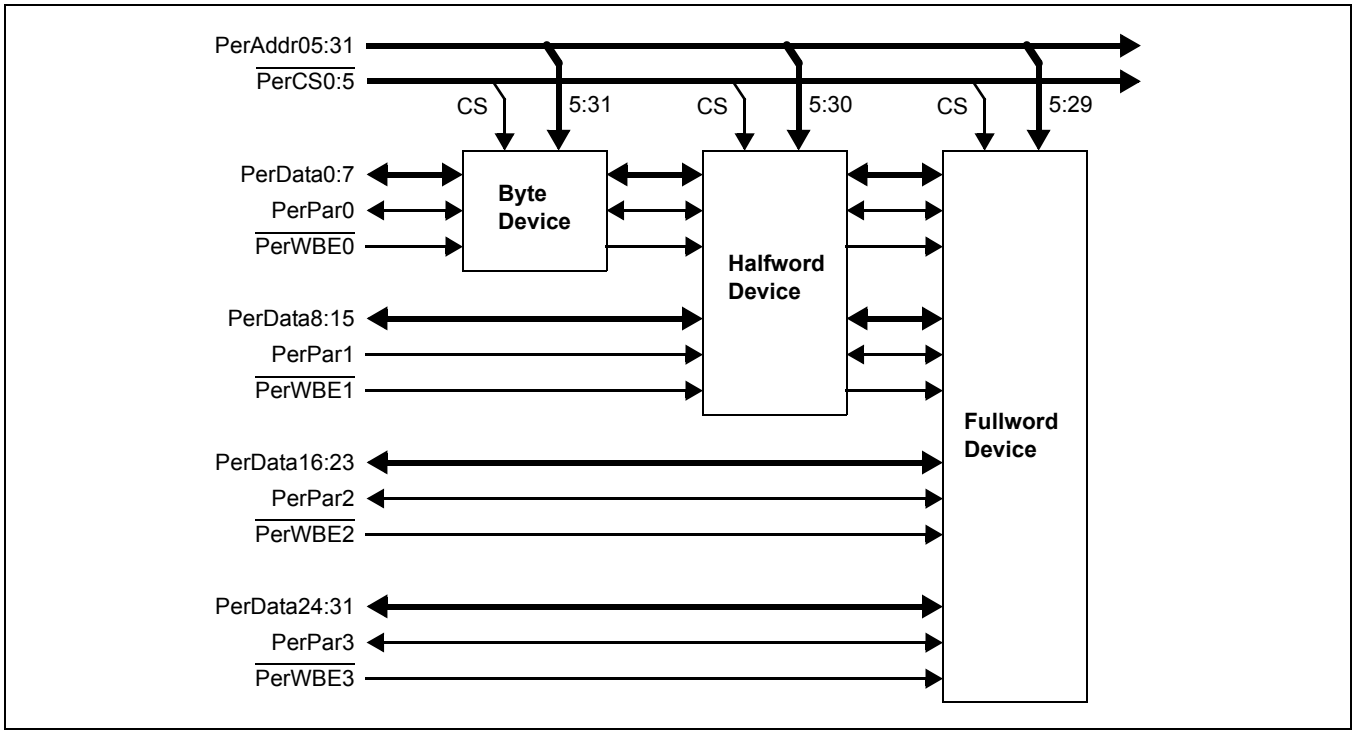
25.1.1 Interfacing to Byte, Halfword, and Word Devices

Figure 25-2 illustrates how to interface from byte, halfword, and word devices to the peripheral data bus. When devices are connected in this way, the EBC supports burst transfers and automatically converts read and write operations to the data width of the external device. As shown in *Figure 25-2*, halfword devices should not connect to PerAddr31. Similarly, a 32-bit device does not require either PerAddr30 or PerAddr31. Instead, the active byte lanes should be inferred from $\overline{\text{PerWBE0:3}}$, the read/write byte enables.

When a large number of byte and halfword devices are attached to the peripheral data bus, the capacitive loading on byte lane 0 (and byte lane 1, if many halfword devices are used) will be much larger than the loading on byte 3, possibly resulting in unacceptable timing performance on byte 0 or byte 1.

If a bank register is configured as word-wide, then byte-wide devices may be attached to the bus in any byte lane (and accessed using byte loads and stores). Similarly, if a bank register is configured as word-wide, then halfword-wide devices may be attached to the bus in the byte 0/byte 1 lane, or in the byte 2/byte 3 lane, and accessed using halfword loads and stores. External logic may be required to develop additional control signals if the data bus is utilized in this manner.

Figure 25-2. Attachment of Devices of Various Widths to the Peripheral Data Bus



25.1.2 Driver Enables

As shown in *Table 25-2*, the output enables for the peripheral address, data, and most of the EBC control signals are configurable.

Pull-ups are also unnecessary on the remainder of the EBC control signals when EBC0_CFG[ATC] = 1, EBC0_CFG[DTC] = 1, and EBC0_CFG[CTC] = 1.

Pull-ups are required if EBC0_CFG[ATC] = 0, EBC0_CFG[DTC] = 0, and EBC0_CFG[CTC] = 0 because if the EBC stops driving PerAddr for a sufficiently long time, PerAddr can charge/discharge to the threshold voltage of the receivers on the bus. This has the potential to cause the receivers to oscillate, creating excessive noise and/or low current draw.

Both chip and system resets set the output enable control bits EBC0_CFG[ATC] = 1, EBC0_CFG[DTC] = 1, and EBC0_CFG[CTC] = 1. In most applications, clearing ATC, DTC, or CTC is not recommended. If EBC0_CFG[CTC] = 0, EBC control signals can transition from the active state to high impedance without first being driven inactive. To prevent this, all peripheral banks must be configured with at least one hold cycle, EBC0_BnAP[TH] > 0.

Table 25-2. Effect of Driver Enable Programming on EBC Signal States

EBC Operation	ExtReset PerClk	PerCS0:5	PerAddr05:31	PerR/W PerWBE0:3 PerOE PerBLast	PerData0:31 PerPar0:3
Reset[1]	Driven	Driven	Driven	Driven	Driven
Idle	Driven	EBC0_CFG[CTC][2]	EBC0_CFG[ATC][2]	EBC0_CFG[CTC][2]	EBC0_CFG[DTC][2]
Read	Driven	EBC0_BnAP[OEO][2]	EBC0_BnAP[OEO][2]	EBC0_BnAP[OEO][2]	High impedance
Write	Driven	EBC0_BnAP[OEO][2]	EBC0_BnAP[OEO][2]	EBC0_BnAP[OEO][2]	EBC0_BnAP[OEO, OEN][2]
DMA peripheral transfer	Driven	Driven	High impedance	Driven	Driven by DMA Controller

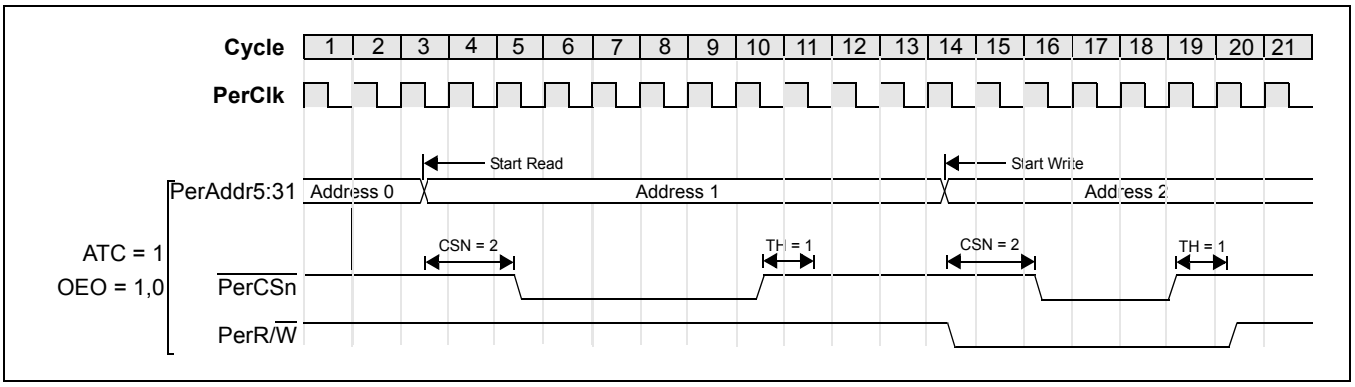
Notes:
1. SysReset active.
2 If the EBC0_CFG bit is set, the signal is driven to the appropriate state during the indicated EBC operation. Otherwise, the I/O is high impedance.
3. The OEO has an effect on driver enables as follows:
   For PerData0:31 and PerPar0:3,
   - If EBC0_CFG[OEO] = 1, data is driven coincident with address for writes.
   - If EBC0_CFG[OEO] = 0, data is driven EBC0_BnAP[OEN] cycles after PerCSn is asserted.

   For PerCS0:5, PerAddr05:31, PerR/W, PerWBE0:3, PerOE, and PerBLast
   - If EBC0_CFG[OEO] = 1, they are driven beginning late in the cycle prior to the address.
   - If EBC0_CFG[OEO] = 0, they are driven coincident with the address.

25.1.2.1 Effect of ATC and OEO On Address Bus Driver

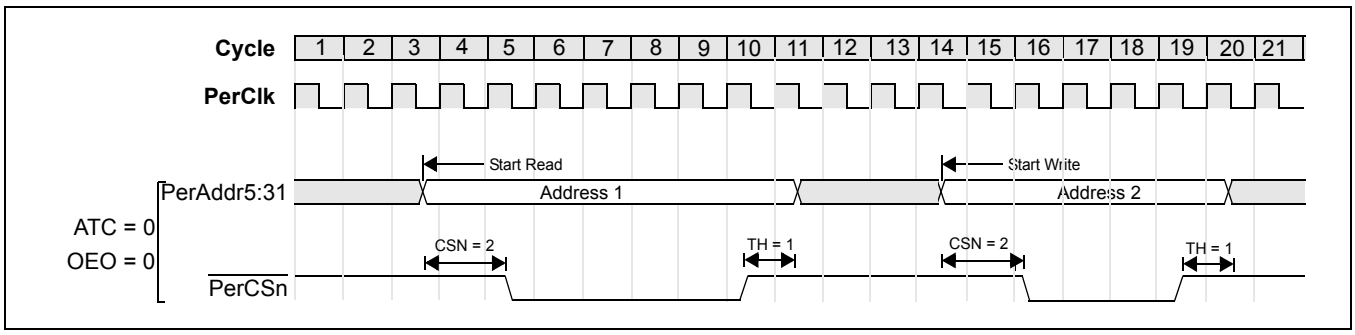
If ATC = 1 and OEO = 0,1 as described in *Figure 25-3*, then the external bus is always driven when the EBC is idle.

Figure 25-3. ATC = 1, OEO = 0, 1



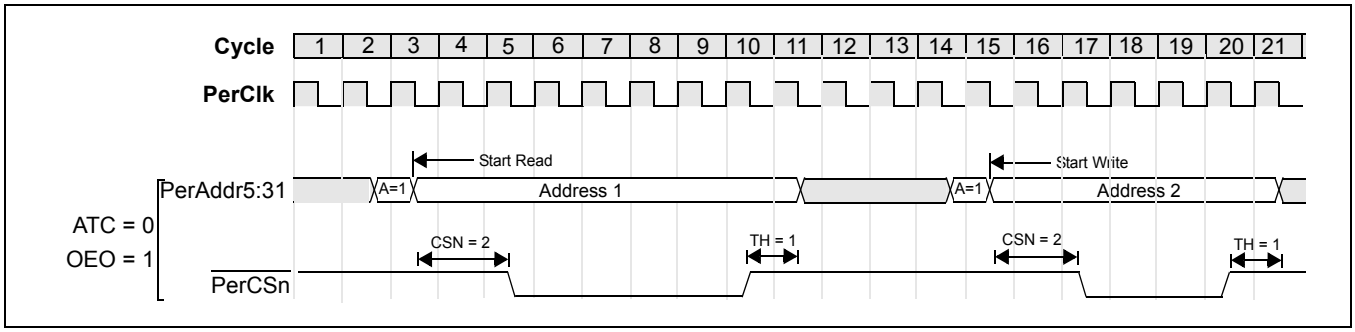
If ATC = 0 and OEO = 0, as described in *Figure 25-4*, then the external bus address driver enables goes active when the cycle starts (off clock edge of first cycle) on the external bus.

Figure 25-4. ATC = 0, OEO = 0



If ATC = 0 and OEO = 1, as described in *Figure 25-5*, the external bus address driver enable goes active prior to the first clock of the cycle on the external bus.

Figure 25-5. ATC = 0, OEO = 1

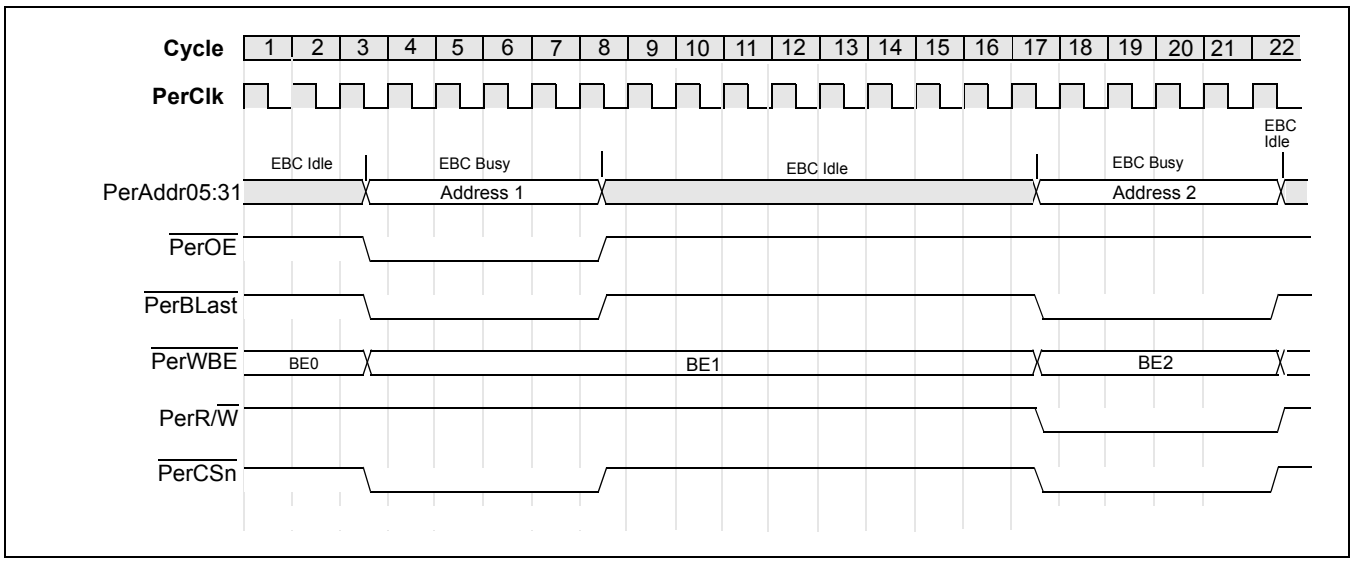


25.1.2.2 CTC and OEO Effect on Control Signal Drivers

The control signals include $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$, $\overline{\text{PerBLast}}$, PerR/$\overline{\text{W}}$, and $\overline{\text{PerWBE}}$.

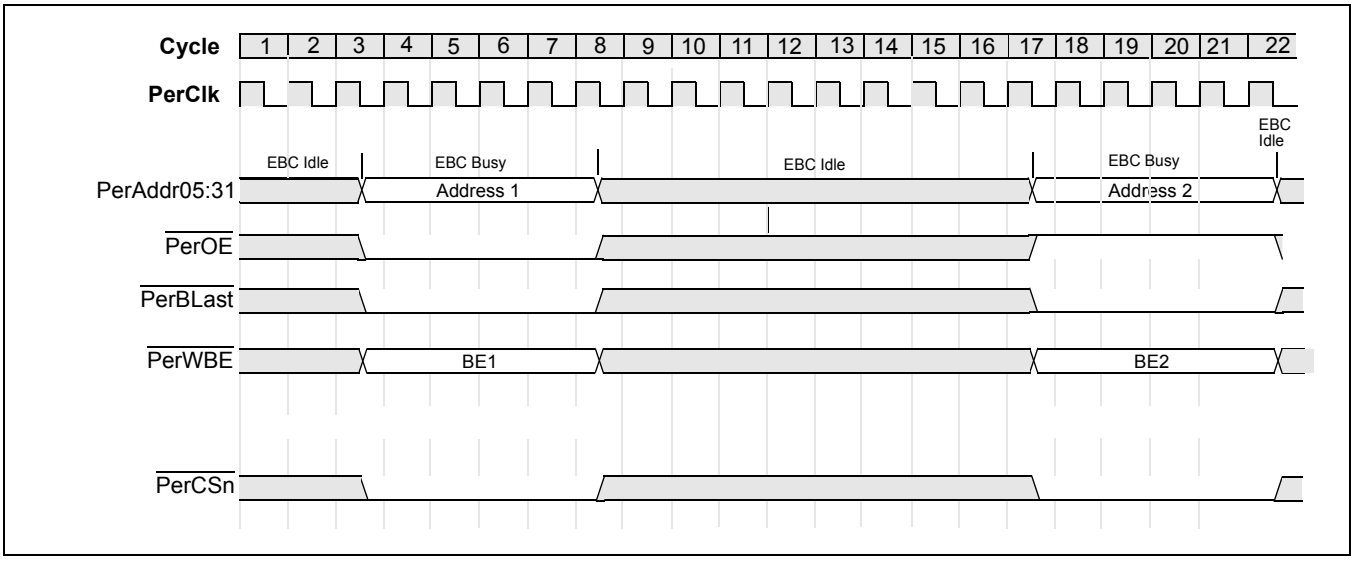
If CTC = 1, as described in *Figure 25-6*, then the control signals are always driven when the EBC is idle.

Figure 25-6. CTC = 1



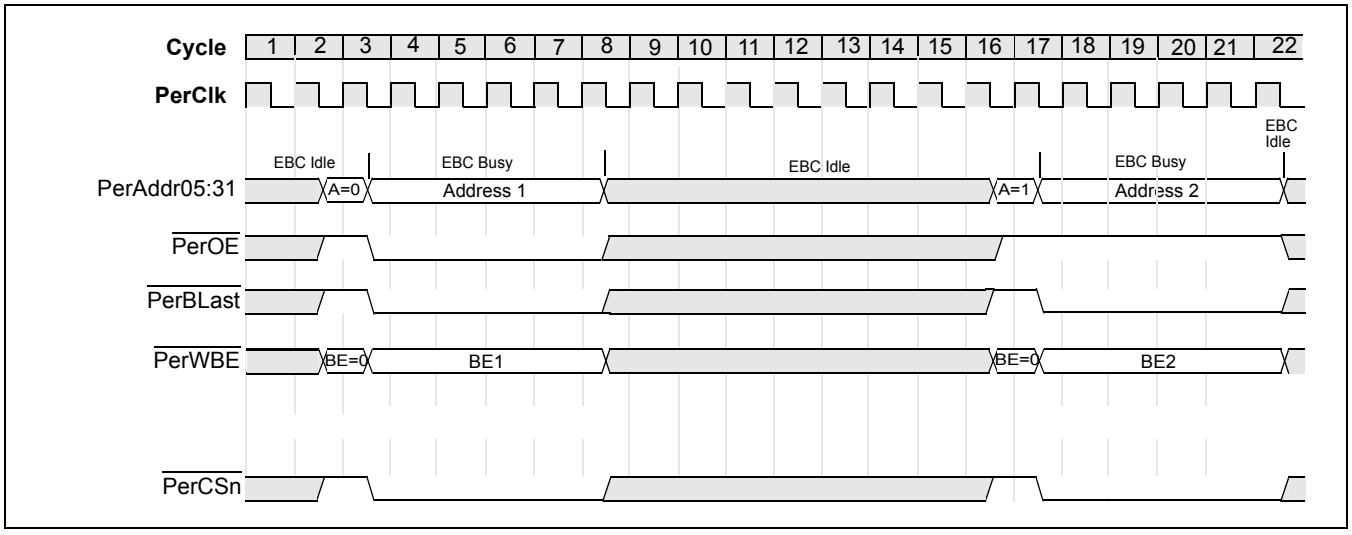
If CTC = 0 and OEO = 0, as described in *Figure 25-7*, then the control signal driver goes active when the cycle starts on the external bus (off clock edge of first cycle on the external bus).

Figure 25-7. CTC = 0, OEO = 0



If CTC = 0 and OEO = 1, as described in *Figure 25-8*, then the control signal drive enable goes active prior to the first clock of the cycle on the external bus.

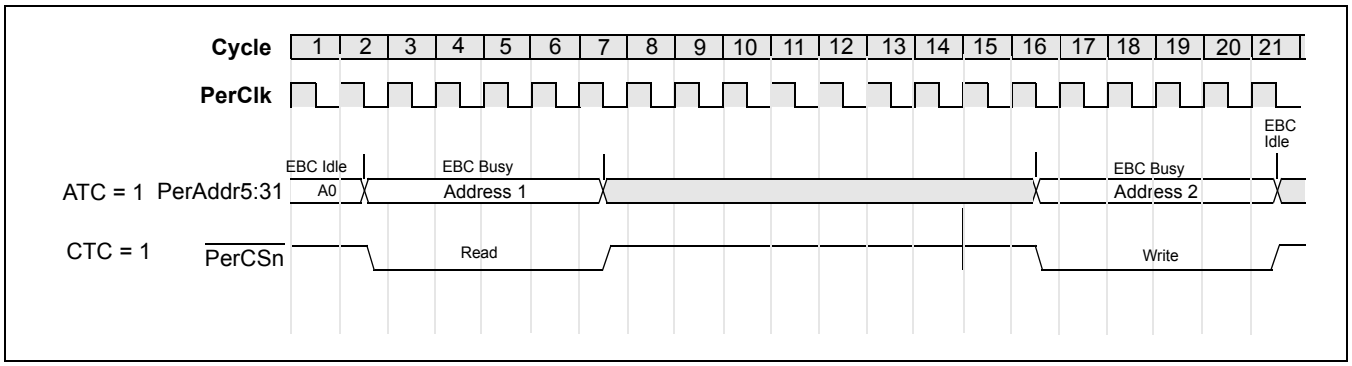
Figure 25-8. CTC = 0, OEO = 1



25.1.2.3 OEN, OEO, and DTC Effect on Bus Enable

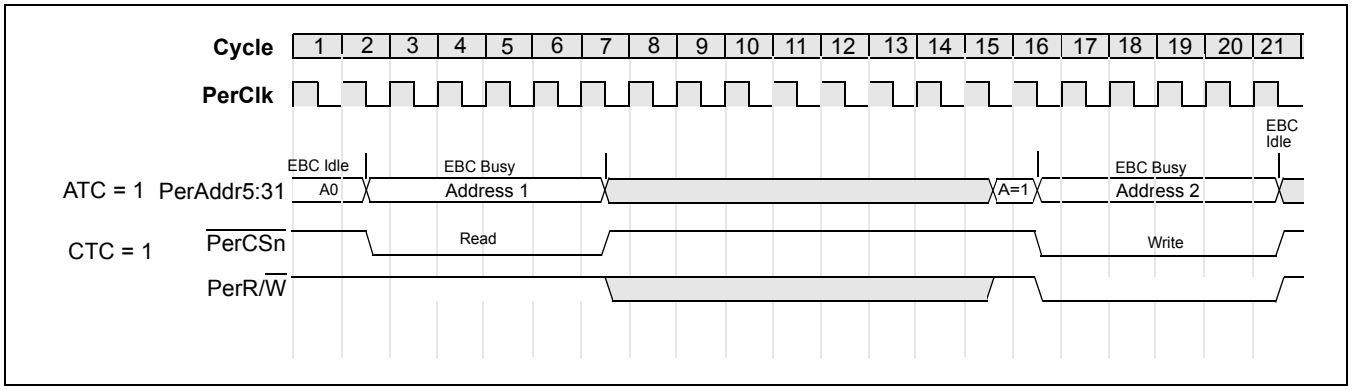
If DTC = 1, as described in *Figure 25-9*, then PerData and PerPar are always driven when the EBC is idle. OEO has no effect when DTC = 1.

Figure 25-9. DTC = 1



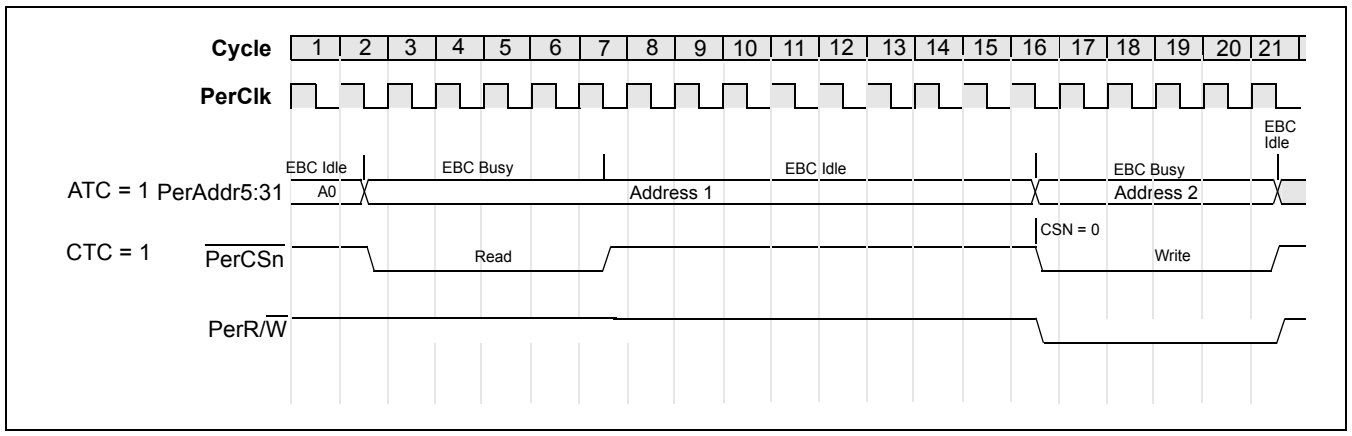
If DTC = 0 and OEO = 1, as described in *Figure 25-10*then the external data bus driver enable goes active at the first clock of the cycle on the external bus.

Figure 25-10. DTC = 0, OEO = 1



If DTC = 0 and OEO = 0, as described in *Figure 25-11*, then the external data bus driver enable is controlled on a bank-by-bank basis by the EBC0_BnAP[OEN] parameter.

Figure 25-11. DTC = 0, OEO = 0



25.2 Non-Burst Peripheral Bus Transactions

The timing of the $\overline{\text{PerCSn}}$, $\overline{\text{PerOE}}$, and $\overline{\text{PerWBE0:3}}$ signals is programmable via the Peripheral Bank Access Parameter (EBC0_BnAP) registers. For non-burst transfers, the access parameter registers control the peripheral bus timing as follows:

- $\overline{\text{PerCSn}}$ becomes active 0-3 PerClk cycles (EBC0_BnAP[CSN]) after the address is driven.
- $\overline{\text{PerOE}}$ is driven low 0-3 PerClk cycles (EBC0_BnAP[OEN]) after $\overline{\text{PerCSn}}$ is active.
- $\overline{\text{PerBLast}}$ is active throughout the entire transfer and is driven high during the programmed hold time (EBC0_BnAP[TH]).
- $\overline{\text{PerWBE0:3}}$ can be either write byte enables or read and write enables.

  If EBC0_BnAP[BEM] = 0, $\overline{\text{PerWBE0:3}}$ are write byte enables and:

  - $\overline{\text{PerWBE0:3}}$ goes active 0-3 (EBC0_BnAP[WBN]) PerClk cycles after $\overline{\text{PerCSn}}$ becomes active.

- PerWBE0:3 becomes inactive 0-3 (EBC0_BnAP[WBF]) PerClk cycles before PerCSn becomes inactive.

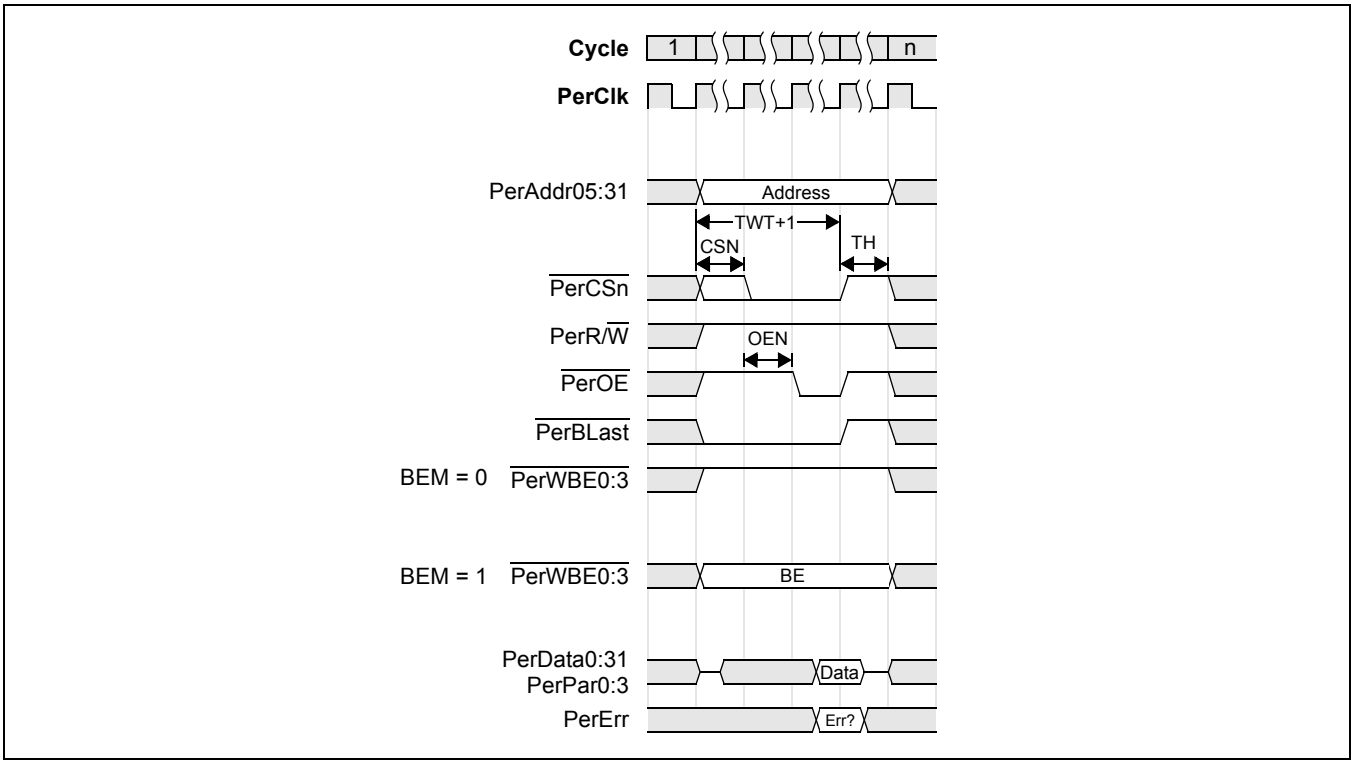
If EBC0_BnAP[BEM] = 1, PerWBE0:3 are read/write byte enables and have timing identical to the peripheral address bus. In this case the EBC0_BnAP[WBN] and EBC0_BnAP[WBF] parameters are ignored.

- 1–256 PerClk cycles (EBC0_BnAP[TWT] + 1) after the address became valid:
  - If EBC0_CFG[CTC] = 1 or EBC0_BnAP[TH] > 0, PerCSn is driven high.
  - If EBC0_CFG[CTC] = 0 and EBC0_BnAP[TH] = 0, PerCSn transitions directly from logic 0 to the high-impedance state.
- The fields EBC0_BnAP[TWT, CSN, OEN, WBN, WBF] in are not independent. For non-burst configured banks (EBC0_BnAP[BME] = 0) that are also non-device-paced (EBC0_BnAP[RE] = 0), it is required that TWT ≥ CSN + MAX(EBC0_BnAP[OEN,WBN]) + EBC0_BnAP[WBF].
- The hold time, EBC0_BnAP[TH], is programmable from 0 to 7 PerClk cycles. During the hold time, the peripheral address bus remains driven with the last address, and all control signals are actively driven high. If the operation was a write, the peripheral data bus continues driving the last data value.
- There is no guarantee of dead cycles between transfers on the peripheral interface for one OPB transfer. If the OPB request size is greater than the bank size or the OPB request size is the same or less than the bank size but with the OPB seqaddr signal asserted, if the EBC does packing data for read and unpacking data for write, and if the number of hold cycles is programmed to 0 (EBC0_BnAP[TH] = 0 and EBC0_BnAP[CSN] = 0), then:
  - PerCSn may not go inactive between the back-to-back transfers.
  - If EBC0_BnAP[OEN] = 0, PerOE may not become inactive between the back-to-back transfers.
  - If EBC0_BnAP[WBN] = 0 and EBC0_BnAP[WBF] = 0, PerWBE0:3 may not go inactive between the back-to-back transfers.
- There will always be a dead cycle between transfers on the peripheral interface that are initiated for separate OPB transfer.

25.2.1 Single Read Transfer

Figure 25-12 shows the peripheral interface timing for a single read transfer from a non-burst enabled (EBC0_BnAP[BME] = 0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM] = 1), the byte enables are also output concurrently on PerWBE0:3. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address, while PerOE goes low EBC0_BnAP[OEN] cycles after PerCSn. The EBC then waits until EBC0_BnAP[TWT] + 1 cycles have elapsed since the start of the transaction and then reads the data bus and the peripheral error input, PerErr. If parity checking is enabled (EBC0_BnAP[PAR] = 1), the parity is also read at this time. The EBC then drives PerCSn, PerOE, and PerBLast high and waits EBC0_BnAP[TH] cycles.

Figure 25-12. Single Read Transfer



25.2.2 Single Write Transfer

Figure 25-13 shows the peripheral interface timing for a single write transfer to a non-burst enabled (EBC0_BnAP[BME] = 0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address. At this point the signalling sequence depends on whether or not byte enable mode is enabled for the bank.

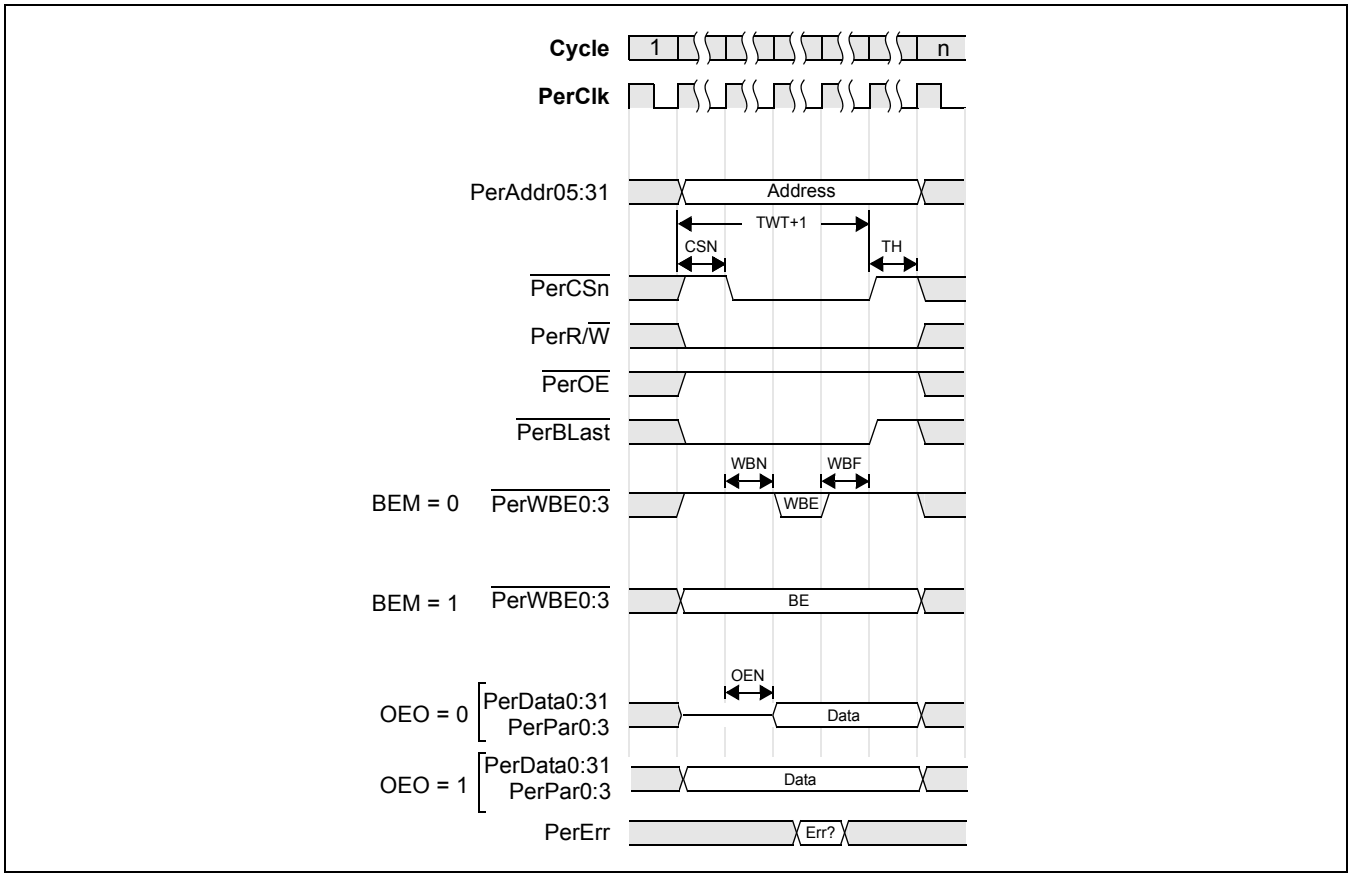
- If EBC0_BnAP[BEM] = 0, byte enable mode is disabled and the PerWBE0:3 are write byte enables. The appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn. The EBC then waits until (EBC0_BnAP[TWT] - EBC0_BnAP[WBF] + 1) cycles have elapsed since the start of the transaction, then drives all the PerWBE0:3 inactive.
- If EBC0_BnAP[BEM] = 1, the PerWBE0:3 lines are byte enables and have the same timing as the peripheral address bus.

OEO also has an effect on PerData and PerPar.

- IF EBC0_CFG[OEO] = 1, PerData and PerPar become actively driven the same time as the address.
- IF EBC0_CFG[OEO] = 0, PerData and PerPar become actively driven EBC0_BnAP[OEN] after PerCSn falls.

After EBC0_BnAP[TWT+1] cycles elapse from the start of transfer, PerCSn and PerBLast are driven high. The EBC then waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

Figure 25-13. Single Write Transfer, EBC0_CFG[OEO] = 0/1



25.3 Burst Transactions

Bursting is controlled on a per-bank basis by the burst mode enable bit in the EBC0_BnAP registers. When enabled (EBC0_BnAP[BME] = 1) this mode activates bursting for all OPB sequential transfers to the EBC, and all packing and unpacking operations.

OPB sequential transfers to the OPB occur when the CPU performs cache line transfers to the EBC or when the CPU or any other PLB master performs fixed length burst transfers to the EBC.

Under many conditions, the EBC will execute extra read transfers on the peripheral interface for a sequential OPB burst operation. For example, if the transfer size on the OPB is less than or equal to the bank size, the EBC will do extra read(s) on the peripheral interface before the OPB termination can be detected. If extra read transfers are not allowed for a specific peripheral device (a FIFO for example), the EBC must be programmed to support fixed length bursts in EBC0_BnAP[BCE]. If the fixed length burst count is complete and the OPB operation has not terminated, the EBC will execute another fixed length burst read. The fixed length burst will terminate early if an OPB termination is detected.

Note: Access to FIFO devices must start and end on a word boundary.

When bursting is enabled:

- $\overline{\text{PerCSn}}$ becomes active (EBC0_BnAP[CSN]) PerClk cycles after the address becomes valid.
- $\overline{\text{PerCSn}}$ is no longer actively driven:
  - 1–32 (EBC0_BnAP[FWT] + 1) PerClk cycles after the address becomes valid when a single transfer occurs to a burst-enabled bank.
  - 1–8 (EBC0_BnAP[BWT] + 1) PerClk cycles after the last address becomes valid during a burst:
    - If EBC0_CFG[CTC] = 1 or EBC0_BnAP[TH] > 0, $\overline{\text{PerCSn}}$ is driven high.
    - If EBC0_CFG[CTC] = 0 and EBC0_BnAP[TH] = 0, $\overline{\text{PerCSn}}$ transitions directly from logic 0 to the high-impedance state.
- During read operations, $\overline{\text{PerOE}}$ is driven low (EBC0_BnAP[OEN]) PerClk cycles after $\overline{\text{PerCSn}}$ is active. $\overline{\text{PerOE}}$ goes inactive when $\overline{\text{PerCSn}}$ goes inactive.
- For bursts, the EBC drives a new address (EBC0_BnAP[FWT] + 1) + $n \times$ (EBC0_BnAP[BWT] + 1) cycles after the start of the transaction, where n = 0, 1, 2,...
- During write operations, the write data is driven concurrent with each address except the first one.
- For the first address:
  - IF EBC0_CFG[OEO] = 1, PerData and PerPar become actively driven the same time as the address.
  - IF EBC0_CFG[OEO] = 0, PerData and PerPar become actively driven EBC0_BnAP[OEN] after $\overline{\text{PerCSn}}$ falls.
- $\overline{\text{PerWBE0:3}}$ can be either write byte enables or read and write enables.

  If EBC0_BnAP[BEM] = 0, $\overline{\text{PerWBE0:3}}$ are write byte enables and:
  - For the first transfer of a burst or a single transfer to a burst enabled bank, the appropriate write byte enables go low (EBC0_BnAP[WBN]) cycles after $\overline{\text{PerCSn}}$ becomes active. The EBC then waits until EBC0_BnAP[FWT – EBC0_BnAP[WBF] + 1 cycles have elapsed since the start of the transaction and drives $\overline{\text{PerWBE0:3}}$ inactive.
  - The remaining transfers of the burst are similar, except that $\overline{\text{PerWBE0:3}}$ becomes active when each new address is driven on the interface. The $\overline{\text{PerWBE0:3}}$ signals remain low for (EBC0_BnAP[BWT] + 1) – EBC0_BnAP[WBN] – EBC0_BnAP[WBF] cycles.

If EBC0_BnAP[BEM] = 1, PerWBE0:3 are byte enables that have timing identical to the peripheral address bus. In this case, EBC0_BnAP[WBN] and EBC0_BnAP[WBF] are ignored.

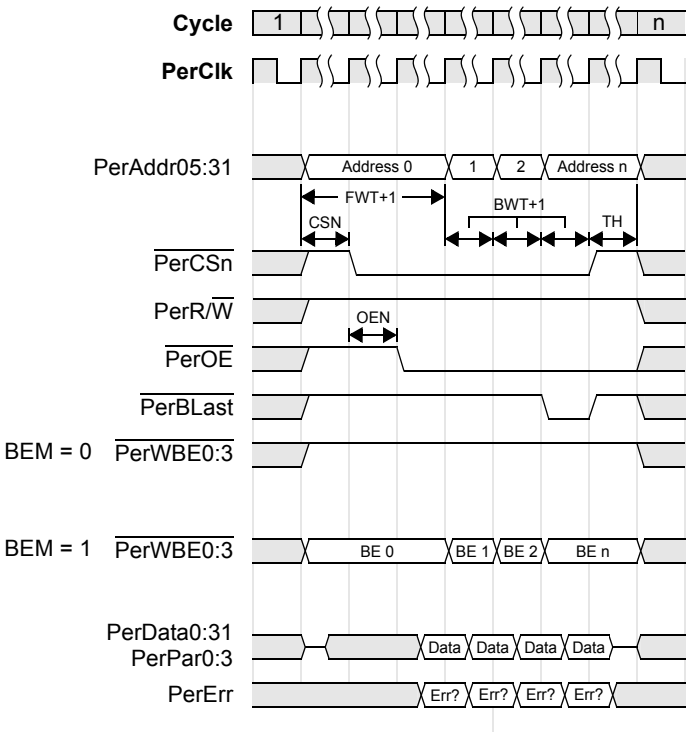
- PerBLast is active throughout the entire last (or only) transfer of a burst operation and is deactivated during the programmed hold time (EBC0_BnAP[TH]).
- EBC0_BnAP[CSN, WBN, OEN] apply to the first (or only) transfer of a burst, while EBC0_BnAP[WBF] applies to all transfers. It is required that FWT ≥ CSN + MAX(EBC0_BnAP[OEN,WBN]) + EBC0_BnAP[WBF], and EBC0_BnAP[BWT] ≥ WBF.
- Hold time (EBC0_BnAP[TH]) is programmable from 0 to 7 cycles. During the hold time, the peripheral address bus remains driven, and all control signals are driven inactive. If the operation was a write, the peripheral data bus continues driving the write data.
- There is no guarantee of dead cycles between transfers on the peripheral interface for one OPB transfer. If the OPB request size is greater than the bank size or the OPB request size is the same or less than the bank size but with the OPB seqaddr signal asserted, if the EBC does packing data for read and unpacking data for write, and if the number of hold cycles is programmed to 0 (EBC0_BnAP[TH] = 0 and EBC0_BnAP[CSN] = 0), then:
  - PerCSn may not go inactive between the back-to-back transfers.
  - If EBC0_BnAP[OEN] = 0, PerOE cannot become inactive between the back-to-back transfers.
  - If EBC0_BnAP[WBN] = 0 and EBC0_BnAP[WBF] = 0, PerWBE0:3 may not go inactive between the back-to-back transfers.
- There will always be a dead cycle between transfers on the peripheral interface that are initiated for separate OPB transfers.

25.3.1 Burst Read Transfer

Figure 25-14 shows the peripheral interface timing for a burst read transfer from a burst enabled (EBC0_BnAP[BME] = 1) bank. The transaction begins with the address being driven. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM] = 1) the byte enables are also output concurrently on PerWBE0:3. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address, while PerOE goes low EBC0_BnAP[OEN] cycles after PerCSn. The EBC waits until EBC0_BnAP[FWT]+1 cycles have elapsed since the start of the transaction and then reads the data bus and the peripheral error input (PerErr). If parity checking is enabled (EBC0_BnAP[PEN] = 1), the parity is read at the same time.

The next address of the burst is then driven, and after EBC0_BnAP[BWT]+1 cycles, the EBC performs the next read. The remaining items in the burst are read in the same manner, except that PerBLast is active during the last data element. The EBC then drives PerCSn, PerOE and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

Figure 25-14. Burst Read Transfer



25.3.2 Burst Write Transfer

Figure 25-15 shows the peripheral interface timing for a burst write transfer to burst enabled (EBC0_BnAP[BME] = 1) bank. The transaction begins with the address being driven. At this point, the signalling sequence depends on whether byte enable mode is enabled for the bank.

- If EBC0_BnAP[BEM] = 0, byte enable mode is disabled, and PerWBE0:3 are write byte enables. In this case, the appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn. The EBC then waits until (EBC0_BnAP[FWT] + 1) – EBC0_BnAP[WBF] cycles have elapsed since the start of the transaction and drives PerWBE0:3 inactive. EBC0_BnAP[WBF] cycles are then allowed to elapse, after which, the address and data are output for the second element in the burst. As shown in *Figure 25-15*, the EBC transfers the subsequent data items in a similar manner.
- If EBC0_BnAP[BEM] = 1, the PerWBE0:3 lines are byte enables and have the same timing as the peripheral address bus. In this configuration, external logic may be necessary to latch write data at the appropriate times. The driving of write data and data parity is controlled by EBC0_BnAP[OEN] when EBC0_CFG[OEO] = 0.

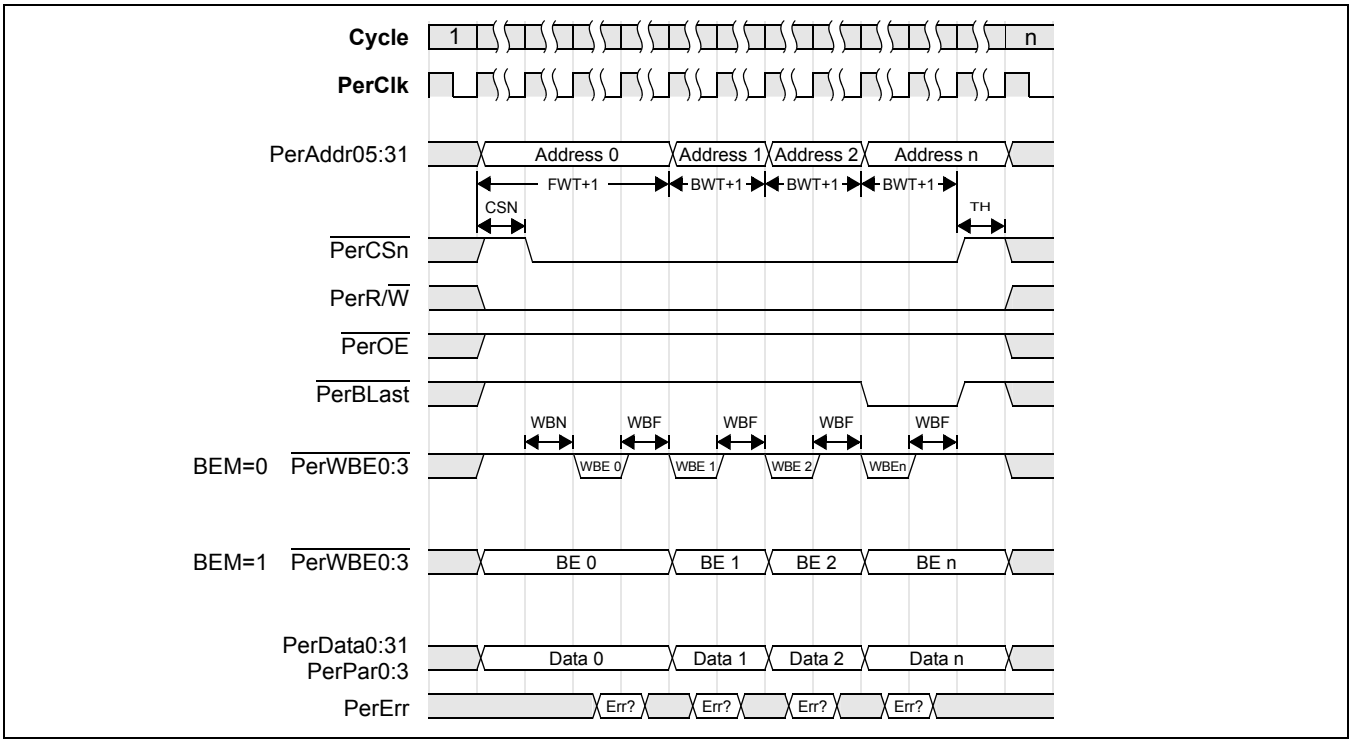
PerBLast goes low at the beginning of the last transfer to indicate that the burst is ending. The EBC then drives PerCSn and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

PerData0:31 and PerPar0:3 depend on OEO and OEN for their first data transfers as follows:

- If EBC0_CFG[OEO] = 1, PerData0:31 and PerPar0:3 become actively driven at the same time as the address.
- If EBC0_CFG[OEO] = 0, PerData0:31 and PerPar0:3 become actively driven EBC0_BnAP[OEN] after PerCSn falls.

PerData0:31 and PerPar0:3 are asserted at the same time as the address.

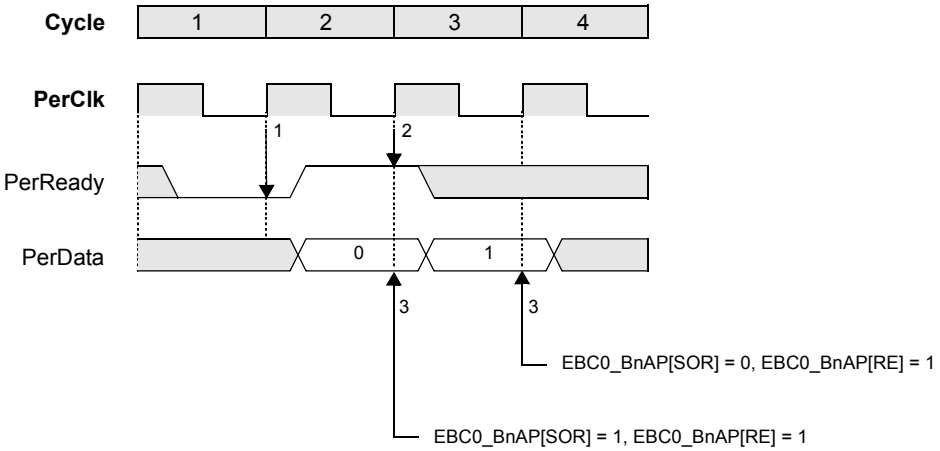
Figure 25-15. Burst Write Transfer



25.4 Device-Paced Transfers

The EBC supports interfacing to peripheral and memory devices with variable timings via device-paced transfers. An EBC bank is configured for device-paced transfers by programming EBC0_BnAP[RE] = 1. During a device-paced transfer, the EBC monitors the ready input (PerReady) and inserts wait states until the external device is ready or a timeout occurs. When using the ready input modes: Sample On Ready enabled and Sample On Ready disabled. The selection of these modes is controlled on a per-bank basis by EBC0_BnAP[SOR]. When Sample On Ready is enabled, (EBC0_BnAP[SOR] = 1) data is transferred on the PerClk rising edge where PerReady is sampled active. When Sampling On Ready is disabled (EBC0_BnAP[SOR] = 0), PerReady sampled active causes the data transfer to occur in the next cycle, which results in an additional cycle of wait time.*Figure 25-16* illustrates available ready modes and latch times transfer.

Figure 25-16. Available Ready Modes and Latch Times



Notes:
1. The above figure assumes that wait time expired in cycle 1 and Ready is being sampled.
2. Arrow indicates Ready is sampled.
3. Arrows indicate data being latched.

- For burst disabled banks (EBC0_BnAP[BME] = 0), sampling of the PerReady input starts EBC0_BnAP[TWT] cycles after the beginning of the transfer. Wait states are inserted and sampling continues once per cycle until either PerReady is high when sampled or a timeout occurs.
- For burst enabled banks (EBC0_BnAP[BME] = 1), sampling of the PerReady input starts EBC0_BnAP[FWT] PerClk cycles after the beginning of the first transfer of a burst and EBC0_BnAP[BWT] cycles after the beginning of subsequent transfers of the burst.
  - If EBC0_BnAP[SOR] = 0, sampling continues every other cycle after the address goes active and the wait period (either TWT for non-burst, or FWT or BWT for burst bank) expires until either PerReady is sampled high or a timeout occurs. If EBC0_BnAP[FWT] = EBC0_BnAP[BWT] = 0, two-cycle transfers occur.
  - If EBC0_BnAP[SOR] = 1, sampling continues once per cycle after the address goes active and the wait period (either TWT for non-burst, or FWT or BWT for burst bank) expires until either PerReady is sampled high or a timeout occurs. If EBC0_BnAP[FWT] = EBC0_BnAP[BWT] = 0, single-cycle transfers occur.
- When EBC0_BnAP[SOR] = 1, data transfer occurs at the same PerClk edge where PerReady is sampled active. In contrast, if EBC0_BnAP[SOR] = 0, the data transfer occurs in the next cycle.
- When EBC0_BnAP[SOR] = 1, if the hold time is set to zero (EBC0_BnAP[TH] = 0), the programmed hold time is ignored, and the EBC performs the transaction with one hold cycle.
- When EBC0_BnAP[RE] = 1, the write byte enable off parameter must be programmed to zero (EBC0_BnAP[WBF] = 0). The Write Byte Enables (/PerWBE0:3) are not deasserted between data transfers but change as needed with the address.

The EBC may be programmed to wait only a limited time for PerReady to become active, or it may be programmed for unlimited wait. If EBC0_CFG[PTD] = 1, timeouts are disabled and the EBC waits indefinitely for an active PerReady.

If EBC0_CFG[PTD] = 0, device-paced timeouts are enabled, and the EBC only waits for the number of PerClk cycles programmed in EBC0_CFG[RTC]. The timeout counter is reset whenever the peripheral address changes. In this manner each data element within a device-paced burst transaction is treated separately for the purposes of determining whether a timeout error occurs. If PerReady does not become active before the timeout counter reaches the value programmed into EBC0_CFG[RTC], the transfer is aborted and an error is signalled. See *Error Reporting* on page 873 for details on how timeout errors are logged.

The required relationship between the access bank parameters changes for device-paced banks, which require that the bank timing wait parameters meet the following relationships:

For non-burst configured banks:

EBC0_BnAP[TWT] > EBC0_BnAP[CSN] + MAX(EBC0_BnAP[OEN, WBN]).

For burst configured bank:

For first transfer:

EBC0_BnAP[FWT] > EBC0_BnAP[CSN] + MAX(EBC0_BnAP[OEN, WBN]).

For following transfers:

EBC0_BnAP[BWT] > EBC0_BnAP[WBF]

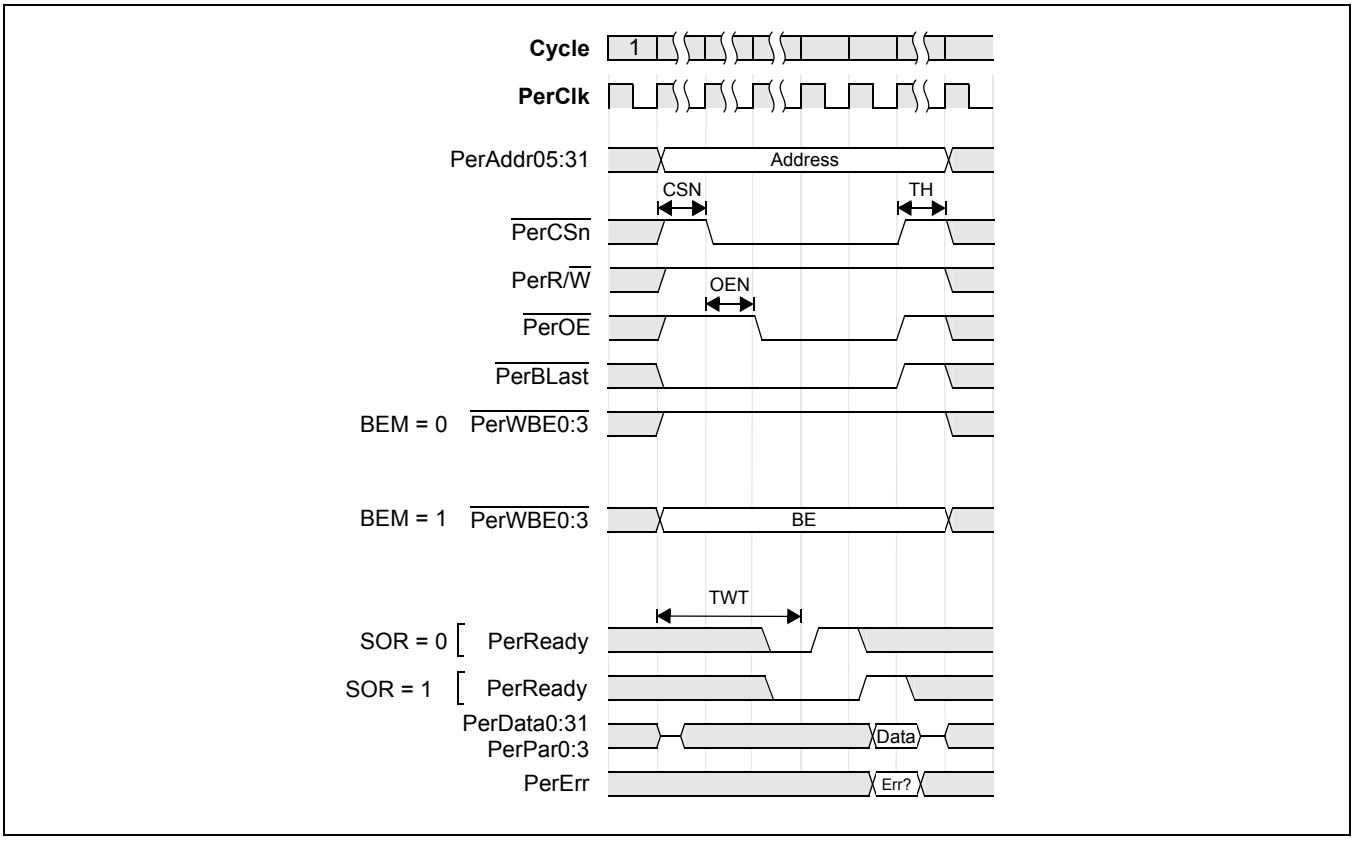
25.4.1 Device-Paced Single Read Transfer

Figure 25-17 shows the peripheral interface timing for a device-paced single read transfer to a burst disabled (EBC0_CFG[BME] = 0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is driven active along with the address. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM] = 1), the byte enables are also output concurrently on PerWBE0:3. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address, while PerOE goes low EBC0_BnAP[OEN] cycles after PerCSn.

The EBC then waits until EBC0_BnAP[TWT] cycles have elapsed since the start of the transaction and then begins sampling PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD] = 1), the EBC waits indefinitely for PerReady to become active; otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

Once PerReady is sampled active and if Sample On Ready is disabled (EBC0_BnAP[SOR] = 0), the EBC waits one more cycle. The EBC then samples the data bus and the peripheral error input (PerErr). If parity checking is enabled (EBC0_BnAP[PEN] = 1), the parity is also read at this time. The EBC then drives PerCSn, PerOE, and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending EBC transfers to occur.

Figure 25-17. Device-Paced Single Read Transfer



25.4.2 Device-Paced Single Write Transfer

Figure 25-18 shows the peripheral interface timing for a device-paced single write transfer to a burst disabled (EBC0_BnAP[BME] = 0) bank. The transaction begins with the address being driven. Since this is a single transfer, PerBLast is also driven active along with the address. At this point, the signalling sequence depends on whether byte enable mode is enabled for the particular bank.

- If EBC0_BnAP[BEM] = 0, byte enable mode is disabled and the PerWBE0:3 are write byte enables. The appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn went low. PerWBE0:3 return high on the same PerClk edge that the write data is transferred (see below).
- If EBC0_BnAP[BEM] = 1, the PerWBE0:3 lines are byte enables and have the same timing as the peripheral address bus.

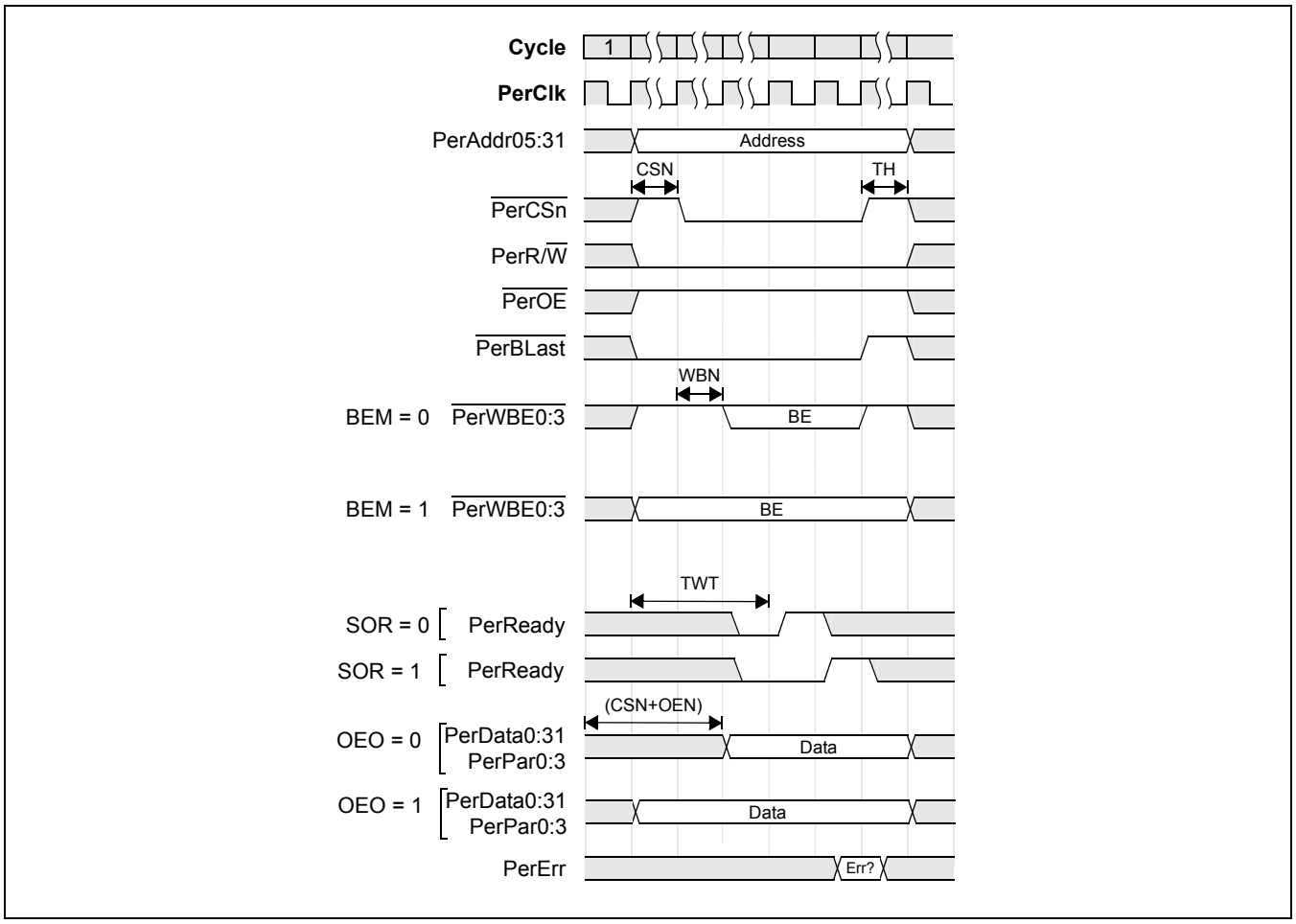
EBC0_CFG[OEO] also has an effect on PerData and PerPar as follows:

- If EBC0_CFG[OEO] = 1, PerData and PerPar become actively driven at the same time as the address.
- If EBC0_CFG[OEO] = 0, PerData and PerPar become actively driven EBC0_BnAP[OEN] cycles after PerCSn falls.

The EBC then waits until EBC0_BnAP[TWT] cycles have elapsed since the start of the transaction and then begins sample PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD] = 1), the EBC waits indefinitely for PerReady to become active. Otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR] = 0), the EBC waits one more cycle. At this point, the write transfer occurs, and the EBC reads the peripheral error input (PerErr). The EBC then drives PerCSn, PerOE, and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending EBC transfers to occur.

Figure 25-18. Device-Paced Single Write Transfer



25.4.3 Device-Paced Burst Read Transfer

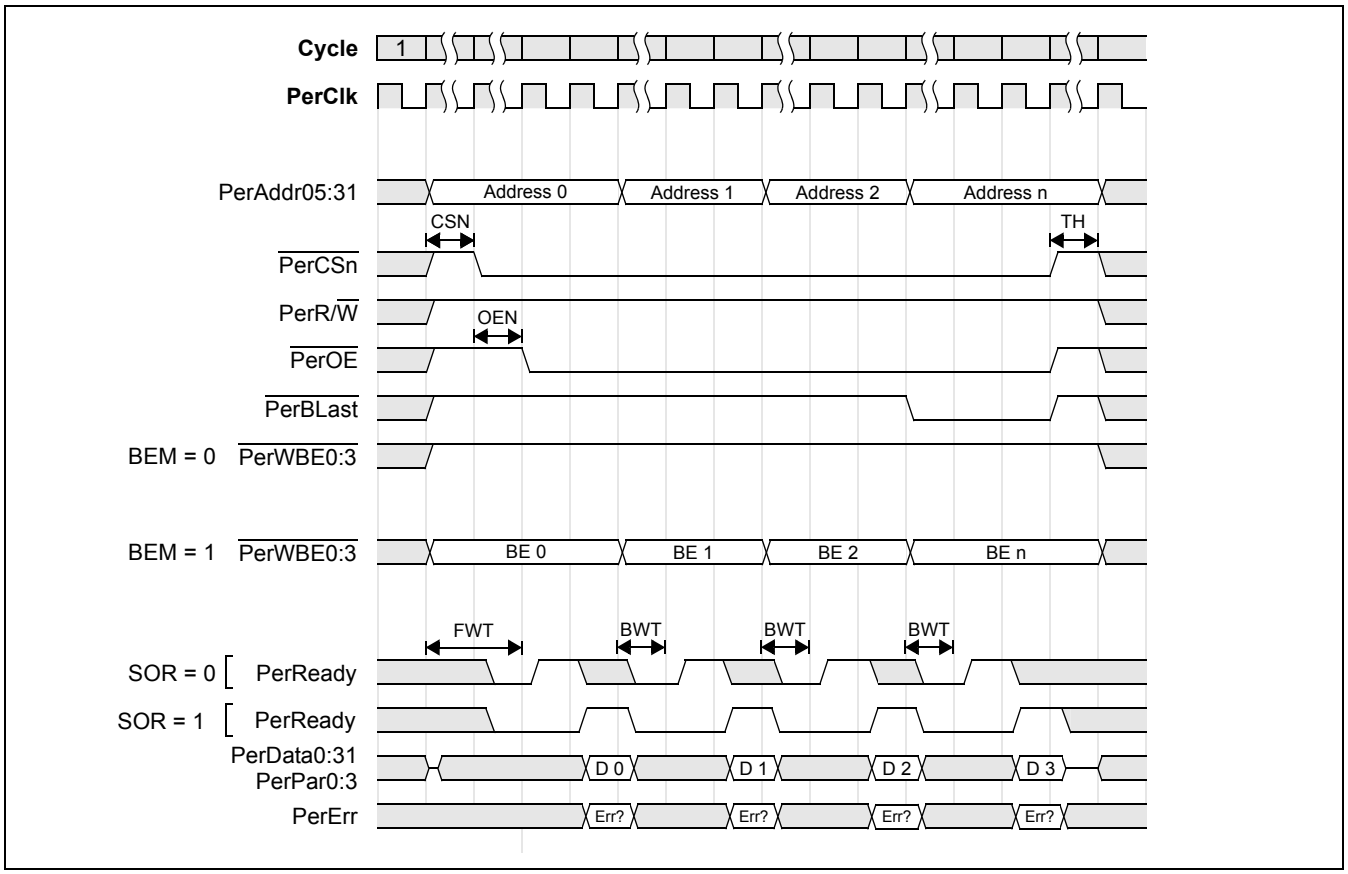
Figure 25-19 shows the peripheral interface timing for a device-paced burst read transfer from a burst enabled (EBC0_BnAP[BME] = 1) bank. The transaction begins with the address being driven. If byte enable mode is enabled for the bank (EBC0_BnAP[BEM] = 1), the byte enables are also output concurrently on PerWBE0:3. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address, while PerOE goes low EBC0_BnAP[OEN] cycles after PerCSn. The EBC then waits until EBC0_BnAP[FWT] cycles have elapsed since the start of the transaction and begins sampling PerReady.

If device-paced timeouts are disabled (EBC0_CFG[PTD] = 1), the EBC waits indefinitely for PerReady to become active; otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of each new address until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR] = 0), the EBC waits one more cycle before sampling read data. The EBC then reads the data bus and the peripheral error input (PerErr). If parity checking is enabled (EBC0_BnAP[PEN] = 1), the parity is also read. See *Burst Transactions* on page 858 for information on extra reads during burst operations.

The next address of the burst is then driven, and after EBC0_BnAP[BWT] cycles, the EBC waits for PerReady as described above. The remaining items in the burst are read in this same manner, except that PerBLast is active during the last data element. The EBC then drives PerCSn, PerOE, and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

Figure 25-19. Device-Paced Burst Read Transfer



25.4.4 Device-Paced Burst Write Transfer

Figure 25-20 shows the peripheral interface timing for a device-paced burst write transfer to a burst enabled (EBC0_BnAP[BME] = 1) bank. The transaction begins with the address being driven. PerCSn then becomes active EBC0_BnAP[CSN] cycles after the address. At this point, the signalling sequence depends on whether or not byte enable mode is enabled for the bank.

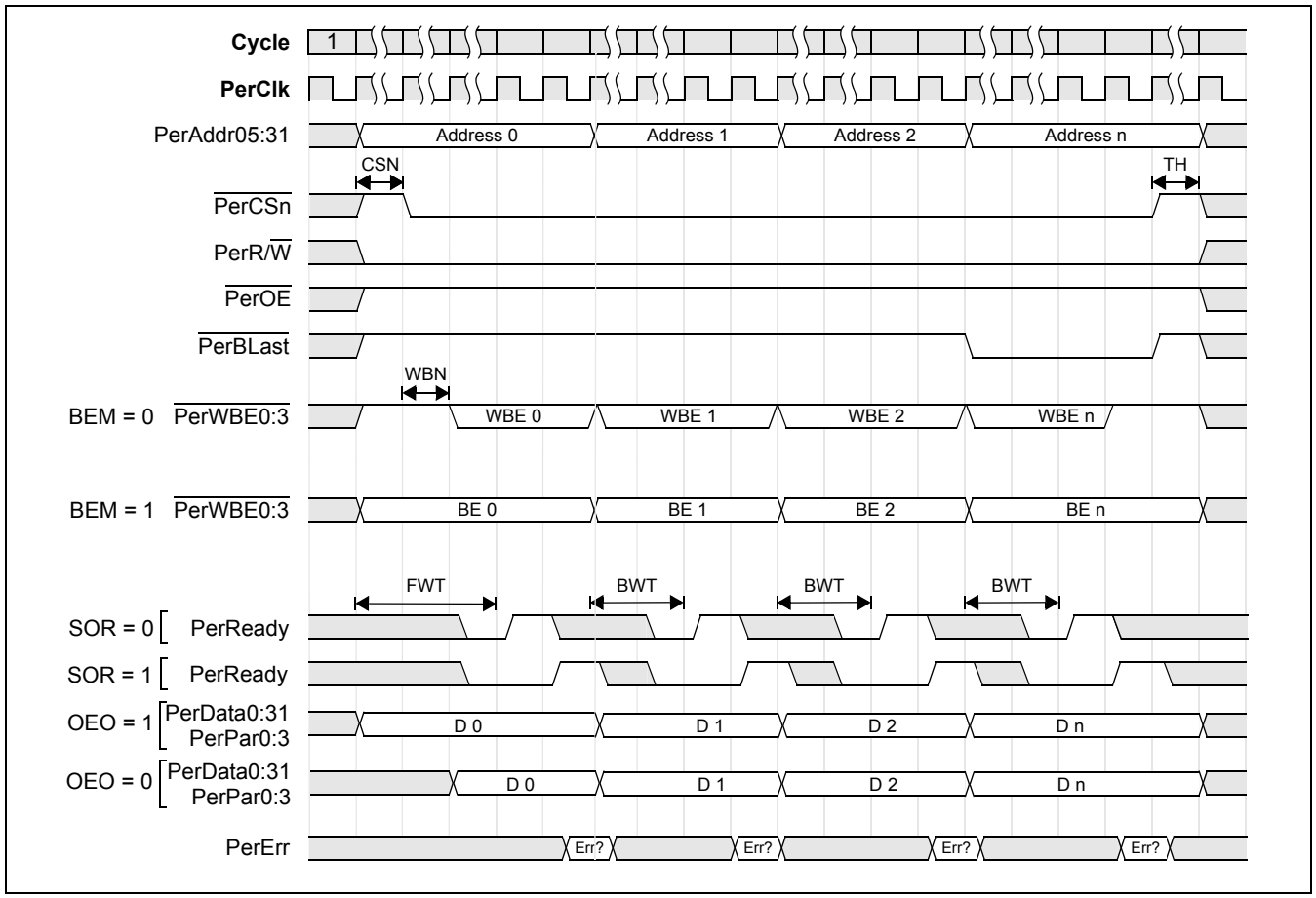
- If byte enable mode is disabled (EBC0_BnAP[BEM] = 0), PerWBE0:3 are write byte enables. In this case, the appropriate write byte enables go low EBC0_BnAP[WBN] cycles after PerCSn becomes active for the first element in a burst and at the same time as each new address for the remainder of the burst.
- If EBC0_BnAP[BEM] = 1, PerWBE0:3 are byte enables and have the same timing as PerAddr05:31.

The EBC then waits until EBC0_BnAP[FWT] cycles have elapsed since the start of the transaction and begins sampling PerReady. If device-paced timeouts are disabled (EBC0_CFG[PTD] = 1), the EBC waits indefinitely for PerReady; otherwise, the EBC waits only EBC0_CFG[RTC] cycles from the start of the transaction until logging a timeout error.

If PerReady is sampled active and Sample On Ready is disabled (EBC0_BnAP[SOR] = 0), the EBC waits one more cycle. At this point, the write transfer occurs, and the EBC reads the peripheral error input (PerErr).

The next address of the burst is then driven, and after EBC0_BnAP[BWT] cycles, the EBC waits for PerReady as described above. The remaining items in the burst are transferred in this same manner, except that PerBLast is active for the last data element. The EBC then drives PerCSn, PerOE, and PerBLast high and waits EBC0_BnAP[TH] cycles before allowing any pending transfers to occur.

Figure 25-20. Device-Paced Burst Write Transfer



25.5 EBC Registers

All EBC configuration and status registers are accessed using the **mtdcr** and **mfdcr** instructions. Access to these registers is performed using an indirect addressing method through the EBC0_CFGADDR and EBC0_CFGDATA registers (see *Table 25-3*).

Table 25-3. EBC DCR Access

Mnemonic	Register	DCR Number	Access	Page
EBC0_CFGADDR	EBC Address Register	0x012	R/W	869
EBC0_CFGDATA	EBC Data Register	0x013	R/W	869

Table 25-4 lists the indirectly accessed EBC configuration and status registers.

Table 25-4. EBC Configuration and Status Registers

Mnemonic	Name	Offset	Access	Page
EBC0_B0CR	Peripheral Bank 0 Configuration Register	0x00	R/W	869
EBC0_B1CR	Peripheral Bank 1 Configuration Register	0x01	R/W	869
EBC0_B2CR	Peripheral Bank 2 Configuration Register	0x02	R/W	869
EBC0_B3CR	Peripheral Bank 3 Configuration Register	0x03	R/W	869
EBC0_B4CR	Peripheral Bank 4 Configuration Register	0x04	R/W	869
EBC0_B5CR	Peripheral Bank 5 Configuration Register	0x05	R/W	869
EBC0_B0AP	Peripheral Bank 0 Access Parameters	0x10	R/W	870
EBC0_B1AP	Peripheral Bank 1 Access Parameters	0x11	R/W	870
EBC0_B2AP	Peripheral Bank 2 Access Parameters	0x12	R/W	870
EBC0_B3AP	Peripheral Bank 3 Access Parameters	0x13	R/W	870
EBC0_B4AP	Peripheral Bank 4 Access Parameters	0x14	R/W	870
EBC0_B5AP	Peripheral Bank 5 Access Parameters	0x15	R/W	870
EBC0_BEAR	Bus Error Address Register	0x20	R	874
EBC0_BESR	Peripheral Bus Error Status Register	0x21	R/W	874
EBC0_CFG	External Bus Configuration	0x23	R/W	875
EBC0_CID	EBC ID	0x24	R	877

To access one of these registers, software must first write the address offset into the EBC0_CFGADDR register. The target register can then be read or written through the EBC0_CFGDATA DCR address. The following PowerPC code illustrates this procedure by writing the EBC0_BnCR register and then reading back the written value.

```
li        r3,EBC0_B0CR              ! address offset
lis       r4,<config upper>         ! upper 16-bits of configuration data
ori       r4,r4,<config lower>      ! lower 16-bits of configuration data
mtdcr     EBC0_CFGADDR,r3           ! set offset addr
mtdcr     EBC0_CFGDATA,r4           ! write config data
mfdcr     r5,EBC0_CFGDATA           ! read back config data
```

Register bits marked as Reserved must only be written to their defined reset value.

25.5.1 EBC Address Register (EBC0_CFGADDR)

Figure 25-21 shows the EBC0_CFGADDR bit definitions. This register is written by software to indirectly address the EBC registers.

Figure 25-21. EBC Address Register (EBC0_CFGADDR)

0:25		Reserved	
26:31	DCRA	DCR Address Offset	

25.5.2 EBC Data Register (EBC0_DATA)

This register is a data port written by software after it writes the EBC0_CFGADDR to read or write the other EBC registers.

Figure 25-22. EBC Data Register (EBC0_CFGDATA)

0:31	DCRD	DCR Data Port

25.5.3 Peripheral Bank Configuration Registers (EBC0_B0CR–EBC0_B5CR)

These registers must be configured to enable memory in each respective bank. Boot ROM, if installed, must be attached to bank 0 and must use CS0.

If a boot ROM is present, Bank 0 is automatically configured using the chip strapping pins. When using bootstrap options A, C, D and F after system reset, the EBC0_B0CR[BAS] is loaded with a value of 0xFFE, EBC0_B0CR[BS] is loaded with a value of 0b001, and EBC0_B0CR[BU] is loaded with a value of 0b01. When using bootstrap options B and E after system reset, the EBC0_B0CR[BAS] is loaded with a value of 0xFFC, EBC0_B0CR[BS] is loaded with a value of 0b010, and EBC0_B0CR[BU] is loaded with a value of 0b01. The EBC0_B0CR[BW] is loaded with a value of CCR_CPU_bROM_Size[0:2]. Software may reconfigure EBC0_B0CR later if necessary.

EBC0_BnCR[BAS] sets the base address for a peripheral device. The bank starting address must be a multiple of the bank size programmed in EBC0_BnCR[BS]. EBC0_BnCR[BAS] is compared to bits 0:11 of the address. If the address is within the range of an EBC0_BnCR[BAS] field, the associated bank is enabled for the transaction.

Multiple bank registers must not be programmed with the same base address or as overlapping banks. An attempt to use such overlapping banks is not recorded as an error and may cause contention on the external bus.

EBC0_BnCR[BS] sets the number of bytes which the bank may access, beginning with the base address set in EBC0_BnCR[BAS].

EBC0_BnCR[BU] protects banks of physical devices from read or write accesses.

When a write access is attempted to an address within the range of EBC0_BnCR[BAS], and the bank is designated as read-only, a protection error occurs. Also, when a read access is attempted to an address within the range of EBC0_BnCR[BAS] field, and the bank is designated as write-only, a protection error occurs. The address of the attempted access is logged in EBC0_BEAR and type of error is logged in EBC0_BESR.

EBC0_BnCR[BW] controls the width of region accesses. If devices are attached to the data bus, the EBC automatically packs read data and unpacks write data when a data transfer size mismatch exists.

Figure 25-23. Peripheral Bank Configuration Registers (EBC0_B0CR-EBC0_B5CR)			
0:11	BAS	Base Address Select	Specifies the bank starting address, which must be a multiple of the bank size. When using bootstrap options A, C, D and F, Reset Default, EBC0_B0CR[BAS] = 0XFFE. When using bootstrap options B and E, Reset Default, EBC0_B0CR[BAS] = 0XFFC.
12:14	BS	Bank Size 000 1 MB bank 001 2 MB bank 010 4 MB bank 011 8 MB bank 100 16 MB bank 101 32 MB bank 110 64 MB bank 111 128 MB bank	When using bootstrap options A, C, D and F, Reset Default, EBC0_B0CR[BS] = 0b001. When using bootstrap options B and E, Reset Default, EBC0_B0CR[BS] = 0b010.
15:16	BU	Bank Usage 00 Disabled 01 Read-only 10 Write-only 11 Read/Write	Specifies the type of accesses allowed for the bank. A protect error occurs if a write is attempted to a read-only bank or a read is attempted from a write-only bank. Reset default, EBC0_B0CR[BU] = 0b01.
17:18	BW	Bus Width 00 8-bit bus 01 16-bit bus 10 Reserved 11 32-bit bus	Reset Default, EBC0_B0CR[BW] = SDR0_EBC0[RW].
19:31		Reserved	

25.5.4 Peripheral Bank Access Parameters (EBC0_B0AP–EBC0_B5AP)

These registers contain access parameters for their associated banks.

EBC0_BnAP[BME] controls bursting for external devices and all packing and unpacking operations. If EBC0_BnAP[BME] = 1, bursting is enabled. When bursting is enabled, EBC0_BnAP[CSN, OEN, and FWT] apply only to the first transfer, while EBC0_BnAP[BWT and WBN] apply during all remaining transfers of the burst.

EBC0_BnAP[TWT] specifies the number of wait states taken by each transfer to the bank. The number of cycles from address valid to the deassertion of PerCSn is (1 + EBC0_BnAP[TWT]), where $0 \leq TWT \leq 255$. This field is used for non-burst configured banks (EBC0_BnAP[BME] = 0).

EBC0_BnAP[FWT] specifies the number of wait states to be taken by the first access to the bank during a burst configured bank (EBC0_BnAP[BME] = 1). During a burst, the number of cycles from the first address valid to the second address is (1 + EBC0_BnAP[FWT]), where $0 \leq FWT \leq 31$.

EBC0_BnAP[BWT] specifies the number of wait states to be taken by accesses beyond the first during a burst transfer to a burst enabled bank (EBC0_BnAP[BME] = 1). On burst accesses, except for the last, the number of cycles from address valid to the next valid address on each burst access is (1 + EBC0_BnAP[BWT]), where $0 \leq BWT \leq 7$. On the last burst access, the number of cycles from address valid to the deassertion of PerCSn is (1 + EBC0_BnAP[BWT]), where $0 \leq BWT \leq 7$.

EBC0_BnAP[CSN] specifies the chip select turn on delay relative to the address. PerCSn may turn on coincident with the address or be delayed by 1, 2, or 3 PerClk cycles.

EBC0_BnAP[BCE] controls burst read from a peripheral device so extra reads do not occur. If a bank is enabled for burst and EBC0_BnAP[BCE] = 1, EBC can burst for fixed lengths bursts of the programmed value. If a bank is enabled for burst and EBC0_BnAP[BCE] = 1, EBC runs burst cycles on the external bus interface as long as the internal bus master indicates that the next internal OPB transaction is sequential. Enabling fixed length transfers does not guarantee dead cycles between sequential fixed length transfers.

EBC0_BnAP[BCT] specifies the number of transfers from the peripheral device for a decoded OPB sequential read when EBC0_BnAP[BCE] = 1. If the OPB sequential transfer is longer than the number of transfers programmed, the EBC has enough time to pack the data and continues transferring until the OPB sequential transfer is deasserted. If the OPB sequential transfer is shorter than the number of transfers programmed, the EBC divides the transfer into several fixed length transfers. It then continues to do fixed length transfers until the OPB sequential transfer is deasserted. *Table 25-5* shows the actual number of bytes transferred, depending upon the size of the peripheral device and EBC0_BnAP[BCT].

Table 25-5. Fixed Length Burst Count Transfer Size

EBC0_BnAP[BCT]	EBC0_BnCR[BW]	Number of Bytes Read
0b00	8-bit	2
0b00 0b01	16-bit 8-bit	4
0b00 0b01 0b10	32-bit 16-bit 8-bit	8
0b01 0b10 0b11	32-bit 16-bit 8-bit	16
0b10 0b11	32-bit 16-bit	32
0b11	32-bit	64

EBC0_BnAP[OEN], for read operations, specifies when the output enable signal, $\overline{\text{PerOE}}$, is asserted for read operations relative to the chip select signal. If EBC0_BnAP[OEN] = 0, $\overline{\text{PerOE}}$ is asserted coincident with the chip select. If EBC0_BnAP[OEN] = 1, 2, or 3, $\overline{\text{PerOE}}$ is delayed by 1, 2, or 3 PerClk cycles.

EBC0_BnAP[OEN], for write operations, specifies when the peripheral data bus is driven relative to the chip select signal when EBC0_CFG[OEO] = 0.

EBC0_BnAP[WBN], when EBC0_BnAP[BEM]=0, specifies when the write byte enables, $\overline{\text{PerWBE0:3}}$, are asserted relative to the chip select signal. If EBC0_BnAP[WBN] = 0, $\overline{\text{PerWBE0:3}}$ turns on coincident with the chip select. If EBC0_BnAP[WBN] = 1, 2, or 3, $\overline{\text{PerWBE0:3}}$ is delayed 1, 2, or 3 PerClk cycles from the chip select.

EBC0_BnAP[WBF], when EBC0_BnAP[BEM]=0, specifies when the write byte enables are deasserted relative to the deassertion of the chip select signal. If EBC0_BnAP[WBF] = 0, $\overline{\text{PerWBE0:3}}$ goes high coincident with the chip select signal. If EBC0_BnAP[WBF] = 1, 2, or 3, then $\overline{\text{PerWBE0:3}}$ turns off 1, 2, or 3 PerClk cycles before the turn-off of the chip select signal.

Programming Note: It is an error to set EBC0_BnAP[WBF] > EBC0_BnAP[BWT]; moreover, for device-paced transfers (EBC0_BnAP[RE] = 1), EBC0_BnAP[WBF] must be 0.

EBC0_BnAP[TH] specifies the number of PerClk cycles (0–7) that the peripheral bus is held idle after the deassertion of PerCSn. During these cycles, the address bus and data bus are active and PerR/W is valid. During the hold time, chip select, output enable, and write byte enables are inactive. If Ready Mode is used (EBC0_BnAP[RE] = 1) with sample on ready (EBC0_BnAP[SOR] = 1), EBC0_BnAP[TH] must be set to at least 1.

EBC0_BnAP[RE] controls the use of the PerReady input signal. If EBC0_BnAP[RE] = 0, the PerReady input is ignored and no additional wait states are inserted into bus transactions. If EBC0_BnAP[RE] = 1, the PerReady input is examined after the wait period expires; additional wait states are inserted if the PerReady input is 0. The maximum number of wait states in each new address is determined by the settings of EBC0_CFG[PTD, RTC]. If EBC0_CFG[PTD] = 1, the ready timeout function is disabled, and the PPC460EX/EXr/GT waits indefinitely until PerReady = 1. If EBC0_CFG[PTD] = 1, the PPC460EX/EXr/GT waits the number of cycle indicated by EBC0_CFG[RTC] for PerReady to become active. If PerReady does not become active in the allotted time, the address of the error is logged in EBC0_BEAR and the type of error is captured in EBC0_BESR.

EBC0_BnAP[SOR] controls the location of the data transfer cycle with respect to the PerReady input. If EBC0_BnAP[SOR] = 1 the data transfer occurs on the same PerClk edge that PerReady is sampled active, whereas if EBC0_BnAP[SOR] = 0 the data transfer occurs one cycle later.

EBC0_BnAP[BEM] controls whether PerWBE0:3 is active during writes or for both reads and writes.

EBC0_BnAP[PEN] enables odd parity generation and checking.

Figure 25-24. Peripheral Bank Access Parameters (EBC0_B0AP-EBC0_B5AP)

0	BME	Burst Mode Enable 0 Burst mode disabled 1 Burst mode enabled	
1:8	TWT	Transfer Wait 0 to 255 PerClk cycles	If bursting is disabled, BME = 0, wait states on all non-burst transfers.
1:5	FWT	First Wait 0 to 31 PerClk cycles	If bursting is enabled, BME = 1, number of wait states on the first transfer of a burst.
6:8	BWT	Burst Wait 0 to 7 PerClk cycles	If bursting is enabled, BME = 1, number of wait states on non-first transfers of a burst.
9	BCE	Fixed Length Burst Reads 0 Disabled 1 Enabled	If BCE = 1, all burst read operations consist of exactly BCT transfers. When BCE = 0, burst read operations may read more than the requested amount of data.
10:11	BCT	Fixed Length Burst Count 00 2 transfers 01 4 transfers 10 8 transfers 11 16 transfers	The amount of data transferred depends upon the programmed bank width, EBC0_BnCR[BW].
12:13	CSN	Chip Select On Timing 0 to 3 PerClk cycles.	Number of cycles from peripheral address driven to PerCSn low.
14:15	OEN	Output Enable On TIming 0 to 3 PerClk cycles	Number of cycles from PerCSn low to PerOE low.
16:17	WBN	Write Byte Enable On Timing 0 to 3 PerClk cycles	If BEM=0, number of cycles from PerCSn low to PerWBE0:3 active.
18:19	WBF	Write Byte Enable Off Timing (If bursting and ready pin input are disabled) 0 to 3 PerClk cycles	If BEM = 0 and RE = 0, number of cycles PerWBE0:3 becomes inactive prior to PerCSn inactive.

20:22	TH	Transfer Hold 0 to 7 PerClk cycles	Contains the number of hold cycles inserted at the end of a transfer. Hold cycles insert idle bus cycles between transfers to enable slow peripherals to remove data from the data bus before the next transfer begins.
23	RE	Ready Enable 0 PerReady input is disabled 1 PerReady input is enabled	Set for Device Paced Transfers
24	SOR	Sampling of Ready for Device Paced 0 Ready is asserted by the device one clock before data is ready on the external bus 1 Ready is asserted by the device on the same clock that data is ready on the external bus	
25	BEM	Byte Enable Mode 0 PerWBE0:3 pins are only active on write cycles 1 PerWBE0:3 pins are active for both read and write cycles	
26	PEN	Parity Enable 0 Disable Parity checking 1 Enable Parity checking	The EBC implements odd parity.
27:31		Reserved	

25.5.5 Error Reporting

The EBC monitors four kinds of errors when performing read and write transfers. Of these four, bank protect and external bus errors are always checked, while timeout and read parity error checking must be enabled via EBC0_CFG[PTD] and EBC0_BnAP[PEN]. When an enabled error is detected by the EBC, the error condition is logged into the EBC0_BESR, the address is logged into EBC0_BEAR, and an interrupt is asserted to the system interrupt controller (UIC1_SR[bit 5]) for one PerClk cycle. See *Peripheral Bus Error Status Register (EBC0_BESR)* on page 874 for more detailed error handling information.

25.5.5.1 Protect Error

The Requested read or write operation violates the bank usage programmed in EBC0_BnCR[BU]. For example, in all cases of a write attempt to read-only bank, no external bus activity occurs.

25.5.5.2 External Bus Error

The PerErr input was sampled active during the data transfer cycle of a read or write operation. The associated data is read or written as usual.

25.5.5.3 Timeout Error

This error is possible during memory operations when both PerReady sampling is enabled (EBC0_BnAP[RE] = 1) and device paced timeouts are enabled (EBC0_CFG[PTD] = 0). Whenever the peripheral address bus changes, the EBC begins counting PerClk cycles. If the count reaches the value represented by EBC0_CFG[RTC], a timeout error occurs. Note that timeout errors are not possible during the peripheral portion of DMA transfers.

25.5.5.4 Parity Error

A parity error indicates that the parity calculated for the read data did not match the parity read. Parity generation and checking is enabled for memory operations by setting EBC0_BnAP[PEN] = 1 and for DMA peripheral transfers by programming DMA2P40_CRn[PCE] = 1.

Protect errors, External Bus errors on reads, Timeout errors on reads, and Parity errors all generate an OPB bus error and possibly a PLB bus error. The error is reported to the bus master that initiated the read. The PPC440 CPU handles these errors as a machine check. Other PLB or OPB masters respond to this error by recording the error in their bus error status registers and asserting an interrupt to the UIC.

External Bus errors on writes and Timeout errors on writes generate an OPB bus IRQ error and possibly a PLB bus master IRQ error. The error is reported to the bus master that initiated the write for posted transaction. The master IRQ error for PLB posted transactions is also recorded in the SDR0_MIRQ0 or SDR0_MIRQ1 registers. The PPC440 CPU handles these errors as a machine check. Other PLB or OPB masters respond to this error by recording the error in their bus error status registers and asserting an interrupt to the UIC.

25.5.5.5 Error Locking

EBC0_CFG[HFE] controls the locking of error information in the EBC0_BESR and EBC0_BEAR.

- If EBC0_CFG[HFE] = 0, the contents of the EBC0_BEAR and the EBC0_BESR reflect the most recent error.
- If EBC0_CFG[HFE] = 1, only the first error detected is reported in the EBC0_BESR and EBC0_BEAR. Subsequent errors are not permitted to overwrite the information detailing the first error. When software processes an error, it must clear all EBC0_BESR bits by writing 0 to allow another error to be logged.

25.5.5.6 Peripheral Bus Error Address Register (EBC0_BEAR)

The Peripheral Bus Error Address Register (EBC0_BEAR) is a 32-bit register containing the address of the access where a data bus error occurred. If EBC0_CFG[HFE] = 1, enabling error locking, and the EBC0_BEAR is not already locked, the contents of EBC0_BEAR are locked until the Peripheral Bus Error Status Register (EBC0_BESR) is cleared. The contents of the EBC0_BEAR are accessed indirectly through EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

Precise address capture in the EBC0_BEAR when a parity error occurs only applies to banks with at least one cycle of hold time (EBC0_BnAP[TH] > 0). This is because parity errors are not calculated until the cycle after the data was valid on the external bus, and the EBC0_BEAR is loaded with the address on the bus during that cycle. If the device timings do not include at least one hold cycle, the EBC0_BEAR may capture the next address in a burst or the address of the next transaction.

Figure 25-25. Peripheral Bus Error Address Register (EBC0_BEAR)

0:31	BEA	Bus Error Address	

25.5.5.7 Peripheral Bus Error Status Register (EBC0_BESR)

The Peripheral Bus Error Status Register (EBC0_BESR) records the occurrence and type of errors for transactions attempted on behalf of the OPB master. The OPB master may be the DMA controller or the PLB-to-OPB Bridge on behalf of the CPU, PCI0, or any other master. The contents of EBC0_BESR are accessed indirectly through EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

It is possible to have both a parity error and a bus error during the same data transfer. If this occurs, the bus error is detected first and EBC0_BESR and EBC0_BEAR are updated. In the next cycle after the parity error is detected, and, if error locking is not enabled, the error is logged in EBC0_BESR (Refer to *Error Locking* on page 874).

When an external bus input error is detected during an EBC read operation, the EBC0_BESR[EBE] bit is set. If the error is detected during a DMA read operation, the DMA controller logs the error in its status register and generates an interrupt, if interrupts are enabled. If the error is detected during a PLB master read operation, the error is recorded in the appropriate master's PLB4OPB0_BESR0:1 field on the PLB-to-OPB Bridge, and the master is signalled that an error occurred.

Figure 25-26. Bus Error Status Register (EBC0_BESR)

0	PYE	Parity Error Detected on Read 0 No Parity Error Detected 1 Parity Error Detected	
1	PRE	Bank Protection Error 0 No Protection Error 1 Protection Error	
2	PERR	PerErr 0 No PerErr Detected during EBC transaction. 1 PerErr Detected during EBC transaction.	If PerErr is detected at any other time during a read, errAck is not asserted. ErrAck is not asserted for write operations.
3	ETE	External Bus Timeout Error 0 No External Bus Timeout Error Detected 1 External Bus Timeout Error Detected	See PDT bit in *Figure 25.5.5.8* on page 875 for timeout enable information.
4	RWS	Read/write Error Status 0 Errant operation was a write operation 1 Errant operation was a read operation	
5	SAS	Sequential Address status 0 Sequential Address Not Active during error detection 1 Sequential Address Active during error detection	
6	EINT	Error Interrupt 0 EBC interrupt to UIC1 disabled 1 EBC interrupt to UIC1 enabled	EINT is a logical OR of the four possible EBC errors, EBC0_BESR[PYE,PRE,PERR,ETE]. Any of these errors asserts EINT. When EINT = 1, the EBC signals the universal interrupt controller (UIC) of an EBC error. The interrupt status is recorded by bit 5 of UIC1_SR and UIC1_MSR. To generate an interrupt to the PPC440 CPU, set UIC1_ER[bit 5] = 1. The interrupt handler must check the EBC0_BESR[PYE,PRE,PERR,ETE] bit fields to determine the source of the error.
7	DMAEP	DMA External Peripheral Error 0 No error 1 Error	DMA external peripheral error detected during EBC transaction.
8:31		Reserved	

25.5.5.8 EBC Configuration Register (EBC0_CFG)

The contents of EBC0_CFG are accessed indirectly through the EBC0_CFGADDR and EBC0_CFGDATA registers using the **mfdcr** and **mtdcr** instructions.

Figure 25-27. EBC Configuration Register (EBC0_CFG)			
0	LE	Lock Error 0 Do not lock error 1 Lock error	This bit is used for error locking in the EBC. If this bit is set, then the EBC will latch the first error detected and save it with associated parameters and address. If this bit is not set, then the EBC will continue to latch new errors and associated parameters and addresses that will overwrite previous errors.
1	PTD	Device-Paced Time-out Disable 0 Enabled time-outs 1 Disable time-outs	This bit is valid only when EBC0_BnAP[RE] = 1. If PTD = 1, the EBC waits indefinitely for assertion of PerReady during device-paced accesses.
2:4	RTC	Ready Timeout Count 000 16 PerClk cycles 001 32 PerClk cycles 010 64 PerClk cycles 011 128 PerClk cycles 100 256 PerClk cycles 101 512 PerClk cycles 110 1024 PerClk cycles 111 2048 PerClk cycles	When PTD = 0, the number of clock cycles from the assertion of PerAddr05:31 until a time-out error occurs.
5	ATC	Address Bus High Impedance Control 0 External address bus is high impedance when EBC is idle 1 External address bus drive previous value when EBC is idle and has ownership of the peripheral interface	Default after reset is ATC = 1. For details on how the ATC affects driver enables, See *Driver Enables* on page 849.
6	DTC	Data Bus High Impedance Control 0 External data bus is high impedance when EBC is idle 1 External data bus drive previous write data value when EBC is idle and has ownership of the peripheral interface	Default after reset is DTC = 1. For details on how the DTC affects driver enables, See *Driver Enables* on page 849.
7	CTC	Control Signal High Impedance Control 0 External bus Control Signals are high impedance when EBC is idle 1 External bus Control Signals are driven inactive and held when EBC is idle and has ownership of the peripheral interface	Default after reset is CTC = 1. For details on how the CTC affects driver enables, See *Driver Enables* on page 849.
8	OEO	External Bus Override High Impedance Control 0 External Bus Output Enable Override Disabled 1 External Bus Output Enable Override Enabled	Default after reset is OEO = 1. For details on how the OEO affects driver enables, See *Driver Enables* on page 849.
9:13		Reserved	
14	PME	Power Management Enable 0 Disabled 1 Enabled	
15:19	PMT	Power Management Timer 0-31	The EBC makes a sleep request to the Clock and Power Management unit when PME = 1 and the EBC has been idle for 32*PMT PerClk cycles.
20:21	PR	Pending Request Timer 00 -16 01 - 32 10 - 64 11 - 128	Number of PerClk cycles to wait for a retried OPB operation (see Note) to return before relinquishing external bus ownership back to the external master.
22:31		Reserved	

25.5.6 EBC Core ID Register (EBC0_CID)

This register indicates the core ID. The core ID includes the technology, core number, and library revision number for this core. Writes to this register will be ignored.

Figure 25-28. EBC Core ID Register (EBC0_CID)			
0:11	CNUM	Core Number	324
12:19	CVAR	Core Variation	01
20:31	REV	Revision Number	This value is the IBM SCCS revision number tracked and modified by the core designer.

26. I2O/DMA (HSDMA) Controller

I2O/DMA as implemented in the PPC460EX/EXr/GT provides support for I2O messaging and one DMA engine.

- I2O is responsible for managing Message Frame Address (MFA) FIFOs or queues in memory in response to I2O Register read/writes. The I2O controller is also responsible for transferring Message Frames (I2O pull mode).
- The DMA engine provides the functionality to perform memory to memory moves as well as many commonly used memory operations.

Note: In this chapter, the 460EX/EXr/GT might also be referred to as the IO Processor (IOP), which is attached to a host by means of the PCI Express or the PCI bus.

26.1 Features

I2O features include the following:

- I2O pull and push messaging methods (With the aid of the DMA engine)
- Dynamic message frame size
- Programmable FIFO size (4096 MFAs of 64 bits maximum)
- 64-bit and 32-bit MFA sizes
- Registered MFA prefetch and posting
- 32-bit inbound and outbound doorbell registers
- Four 32-bit scratch pad registers

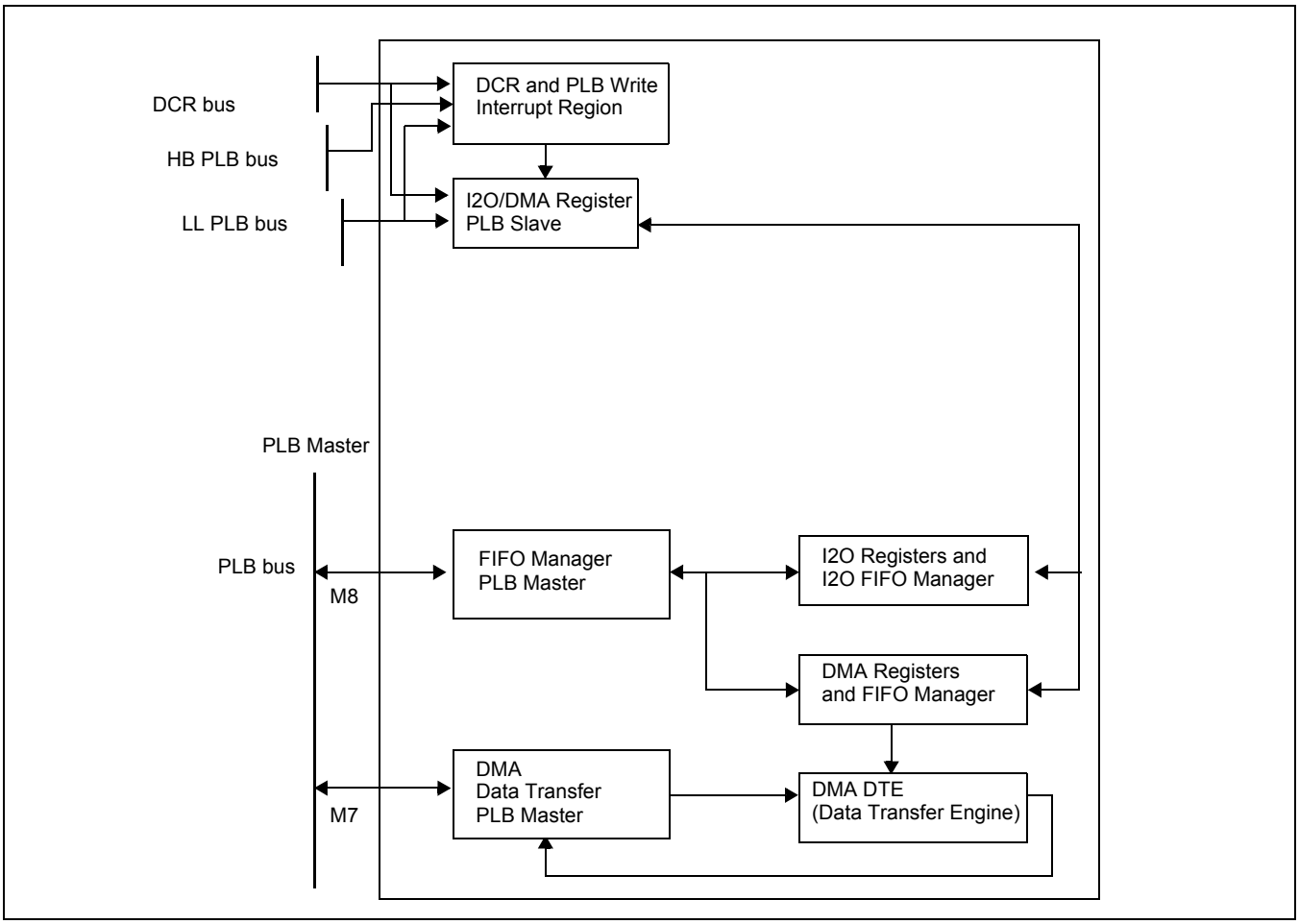
DMA features include:

- Programmable Command Pointer FIFO and Completion FIFO size (up to 2048 DMA operations queued)
- 512-byte buffering
- Simultaneous fill and drain (PLB read/write pipelining)
- Any source PLB address to any destination address
- No memory alignment restrictions on source or destination
- 32-byte command descriptor block
- Maximum transfer size of 16 MB
- Commands include nop, move, multi-cast, data fill, data check, LFSR reset, LFSR fill, LFSR check
- 64-bit addressing
- 1KB buffering
- Prefetch indicators for PCI and PCI Express buffer management

26.2 Functional Overview

The I2O/DMA function is shown in *Figure 26-1*. These modules include Device Control Registers (DCRs) with PLB Write Interrupt Region, one PLB slave (S1 on LL, the low latency bus), two PLB masters (M7 and M8), one I2O Register/FIFO Manager for the DMA port, one DMA Register/FIFO Manager, and one DMA Data Transfer Engine (DTE).

Figure 26-1. I2O/DMA Block Diagram

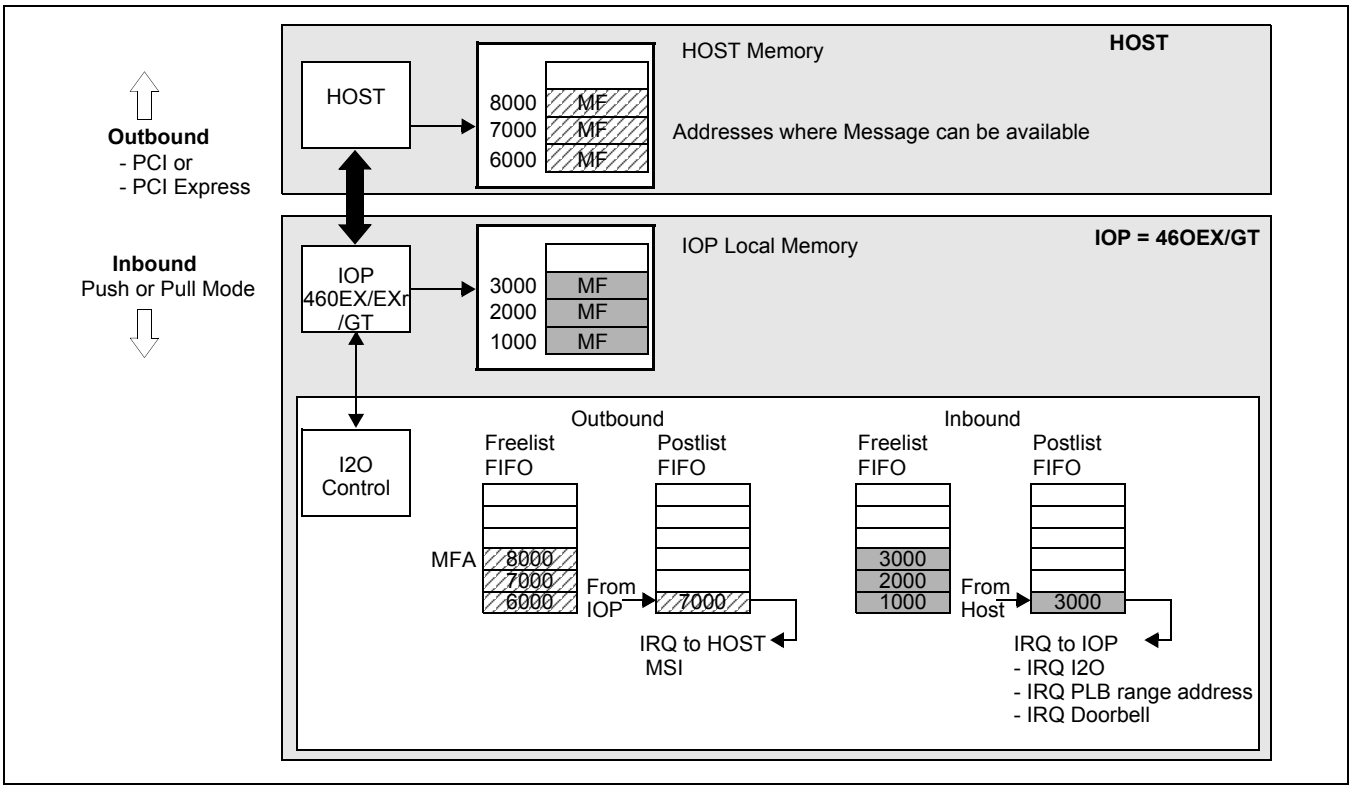


Requests for data transfer are made using either the I2O messaging interface (by means of Message Frames) or direct access to DMA registers. Both I2O pull and push modes are supported. Pointers to commands are used for submission and completion/status.

26.2.1 I2O Inbound Messages

Inbound messages usually travel from the host to the IOP-460EX/EXr/GT. The messages are transported in either "pull" mode or "push" mode. In I2O pull mode, the I2O logic is responsible for transporting the message frame from the host to the 460EX/EXr/GT system memory by means of the DMA engine. In I2O push mode, the host is responsible for transporting the message frame to the 460EX/EXr/GT.

Figure 26-2. I2O Message Principles of Operation



26.2.2 I2O Inbound Pull

In I2O pull mode, the I2O logic is responsible for transporting the message frame from the host to 460EX/EXr/GT system memory by means of the DMA engine. Because the DMA engine is needed, pull mode requires an additional FIFO. This FIFO contains a list of pointers to partially-defined DMA Command Descriptor Blocks (CDBs); see *Table 26-1* for the generic DMA Command Descriptor Block structure. The IOP (PPC440 CPU) fills these CDBs with the necessary DMA operation (MOVE_SG1_SG2), the destination, and the byte count memory space field. This partially completed CDB is referred to as a Free CDB. The 460EX/EXr/GT populates the Free CDB Pointer List FIFO with the starting addresses of every Free CDB created. The 460EX/EXr/GT is responsible for ensuring the Free CDB Pointer List FIFO does not become empty. If it does, the FIFO returns an address of all Fs. The I2O logic will repeatedly request a Free CDB Pointer until a valid pointer is returned.

Table 26-1. Basic DMA Command Descriptor Block (CDB) Structure[a]

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Lower Half of SG3 Address			
0x18	Upper Half of SG3 Address			
0x14	Lower Half of SG2 Address			
0x10	Upper Half of SG2 Address			
0x0C	Reserved	Scatter/Gather (SG) Count		
0x08	Lower Half of SG1 Address			
0x04	Upper Half of SG1 Address			
0x00	DMA OpCode	Attributes	Reserved	

a.A single CDB structure occupies 32 bytes located on a 16-byte boundary. The DMA Data Transfer Engine always fetches 32 bytes when accessing a CDB. The DMA engine only supports the simple addressing Scatter/Gather List (SGL) format defined in I2O (v1.5). No other SGL formats are supported.

To pass messages from the host to the 460EX/EXr/GT, the I2O "framework" must be initialized. This initialization includes:

- MF size count array in the I2O Block. This is an array of values that indicate the number of 16-byte blocks to transfer for an I2O pull operation.
- List of Free CDBs with the destination address (460EX/EXr/GT system memory), DMA Operation, and memory space fields completed.
- Free CDB Pointer List FIFO with an address pointer for each Free CDB in the DMA Command Descriptor Block List.
- Allocation of memory to receive message frames from the host.
- Writes Message Frame Addresses (MFAs) referencing the empty message frames to the Inbound Free List by means of the I2O Register Interface.
- The I2O logic writes the MFA to the memory allocated for the Inbound Free List FIFO in the Inbound Queue.

For a message being transported from the host to the 460EX/EXr/GT by means of pull mode, the following events occur:

- The host creates a Message Frame (MF) in host memory to be communicated to the 460EX/EXr/GT.
- The host writes the Message Frame Address (MFA) to the Inbound Queue Port of the I2O Register Interface.

- The I2O controller decodes the MFA write to the Inbound Queue Port of the I2O Register Interface, detects that Mode = 1 (pull mode), and retrieves the next available Free CDB pointer from the prefetch register of the Free CDB Pointer FIFO. As the Free CDB Pointer FIFO prefetch buffer is emptied, the FIFO control logic will do memory reads to replenish it. To ensure that additional I2O requests are not lost, additional writes to the I2O Register Interface Queue Port are retried until the current I2O transaction is completed.
- The I2O Block writes the MFA (as the source address) and byte count (determined by using the NB field as an index into the Block Count array) values to the memory location specified by the Free CDB Pointer, which completes the required fields of the CDB.
- The I2O controller writes the Free CDB Pointer, now a DMA Command Descriptor Block Pointer (also known as a Command Pointer (CP)) to the DMA Register Interface. The write notifies the DMA engine that a data transfer is required. The I2O controller releases the write blocking on the I2O Register Interface and goes idle (that is, the I2O logic is finished). *Note that the Inbound Queue was not used for this transaction.*

The DMA engine is now responsible for completing the message transfer. The DMA engine is running concurrently with the I2O logic. Once the I2O request is placed in the DMA Command Pointer (CP) FIFO, it will progress through the FIFO until it becomes the first entry in the FIFO. The following events occur once the I2O request has progressed through the FIFO:

- The DMA engine reads the CP for the I2O request from the DMA CP FIFO.
- The DMA engine reads the CDB from the location specified by the CP.
- The DMA engine performs the transfer of the Message Frame from host memory to 460EX/EXr/GT memory.
- The DMA engine places the CP on the DMA Completed Command Descriptor Block Pointer with Status (Completion Status (CS) FIFO). Because the DMA CS FIFO is not empty, an interrupt to the 460EX/EXr/GT is automatically generated.

In response to the 460EX/EXr/GT interrupt, the 460EX/EXr/GT retrieves the CS by doing a DMA Register Interface read. The DMA logic retrieves the CS from memory and returns it to the 460EX/EXr/GT.

If the 460EX/EXr/GT determines that the CS is for a Message Frame, it retrieves and processes the message. If a response is needed, a response is returned by means of the outbound message queue.

26.2.3 I2O Push

The second method for message transport is "push" mode. The first step is to initialize the Inbound Free List FIFO in the Inbound Queue.

- The 460EX/EXr/GT allocates memory to receive message frames from the host.
- The 460EX/EXr/GT writes Message Frame Addresses referencing the empty message frames to the Inbound Free List by means of the I2O Register Interface.
- The I2O logic writes the MFA to memory allocated for the Inbound Free List FIFO in the Inbound Queue.

To send a message to the 460EX/EXr/GT:

- The host builds a Message Frame in host memory.
- The host reads the I2O Register Interface to fetch a Message Frame Address.
- The host writes the message frame to the 460EX/EXr/GT system memory referenced by the MFA.
- The host writes the MFA to the Inbound Post List FIFO by means of the I2O Register Interface.
- The 460EX/EXr/GT is informed the Inbound Post List FIFO is not empty by means of an interrupt. The interrupt is automatically generated whenever the Inbound Post List FIFO is not empty.

The 460EX/EXr/GT responds to the interrupt and reads the I2O Register Interface to extract the MFA from the Inbound Post List FIFO. The 460EX/EXr/GT processes the message and responds appropriately. As the 460EX/EXr/GT processes MFAs from the Inbound Post List FIFO, it replenishes the Inbound Free List FIFO.

26.2.4 Outbound Messages

For outbound messages, there is only one communication mode: push. The first step is to initialize the Outbound Free List FIFO in the Outbound Queue:

- The host allocates memory to receive message frames from the 460EX/EXr/GT.
- The host writes Message Frame Addresses referencing the empty message frames to the Outbound Free List by means of the I2O Register Interface.
- The I2O logic writes the MFA to the memory allocated for the Outbound Free List FIFO in the Outbound Queue.
- To send a message to the host:
- Builds a Message Frame in 460EX/EXr/GT system memory.
- Reads the I2O Register Interface to fetch a Message Frame Address.
- Writes the message frame to the host memory referenced by the MFA or builds a CDB to allow the DMA engine to move the message.
- If the 460EX/EXr/GT uses the DMA engine to move the Message Frame to host memory, the 460EX/EXr/GT writes the CP to the DMA Register Interface; placing it in the DMA CP FIFO.
- If the 460EX/EXr/GT uses the DMA engine to move the Message Frame to host memory, the 460EX/EXr/GT will be interrupted when the DMA engine writes the CS to the DMA Register Interface; placing it in the DMA CS FIFO.
- Writes the MFA to the Outbound Post List FIFO by means of the I2O Register Interface.
- The host is informed that the Outbound Post List FIFO is not empty by means of an interrupt. The interrupt is automatically generated when the Outbound Post List FIFO is not empty.

The host responds to the interrupt and reads the I2O Register Interface to extract the MFA from the Outbound Post List FIFO. The host processes the message and responds appropriately. As the host processes MFAs from the Outbound Post List FIFO, it replenishes the Outbound Free List FIFO.

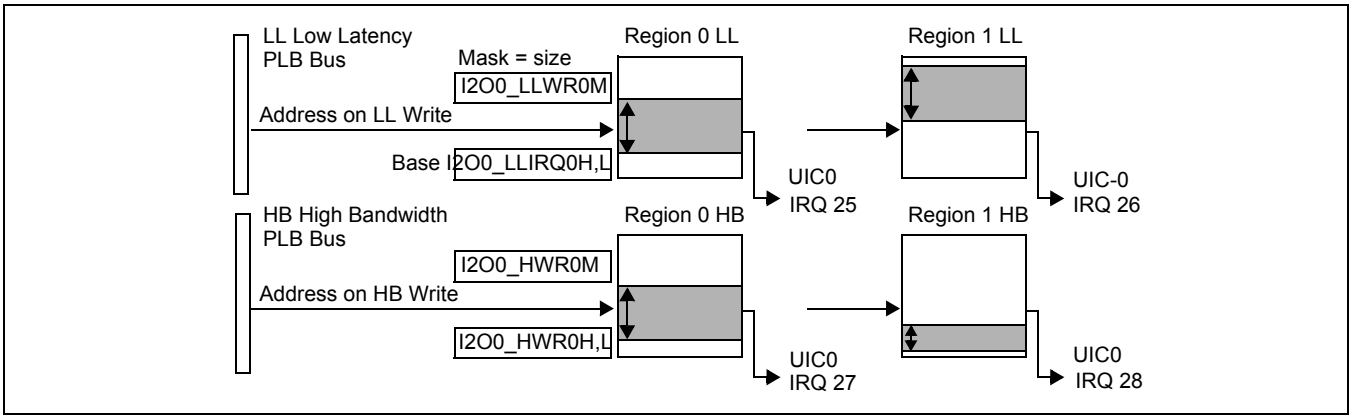
26.2.5 DCR and PLB Write Interrupt Region Module

The I2O/DMA module has two responsibilities: configure the function operation through DCR registers and signal interrupts when a write transaction to a specific address range occurs on either the Low Latency (LL) or High Bandwidth (HB) PLBs (see *I2O/DMA Registers* on page 908 for a full description of the DCR registers available and their addresses).

The Write Interrupt Region logic monitors both PLBs; LL and HB. Two memory-mapped regions may be defined for each PLB (giving a total of four definable regions). To define these regions, the user must program a number of registers that reside in DCR space; see *I2O/DMA Device Control Registers* on page 908. When a slave responds with an address acknowledge to a write within one of the defined regions, this Write Interrupt Region logic send an interrupt pulse to the 460EX/EXr/GT's UIC. The interrupt will remain asserted for a single clock only. A burst write that begins outside the region and ends within the region will not generate an interrupt. Conversely, a burst write starting inside the region and ending outside the region does generate an interrupt.

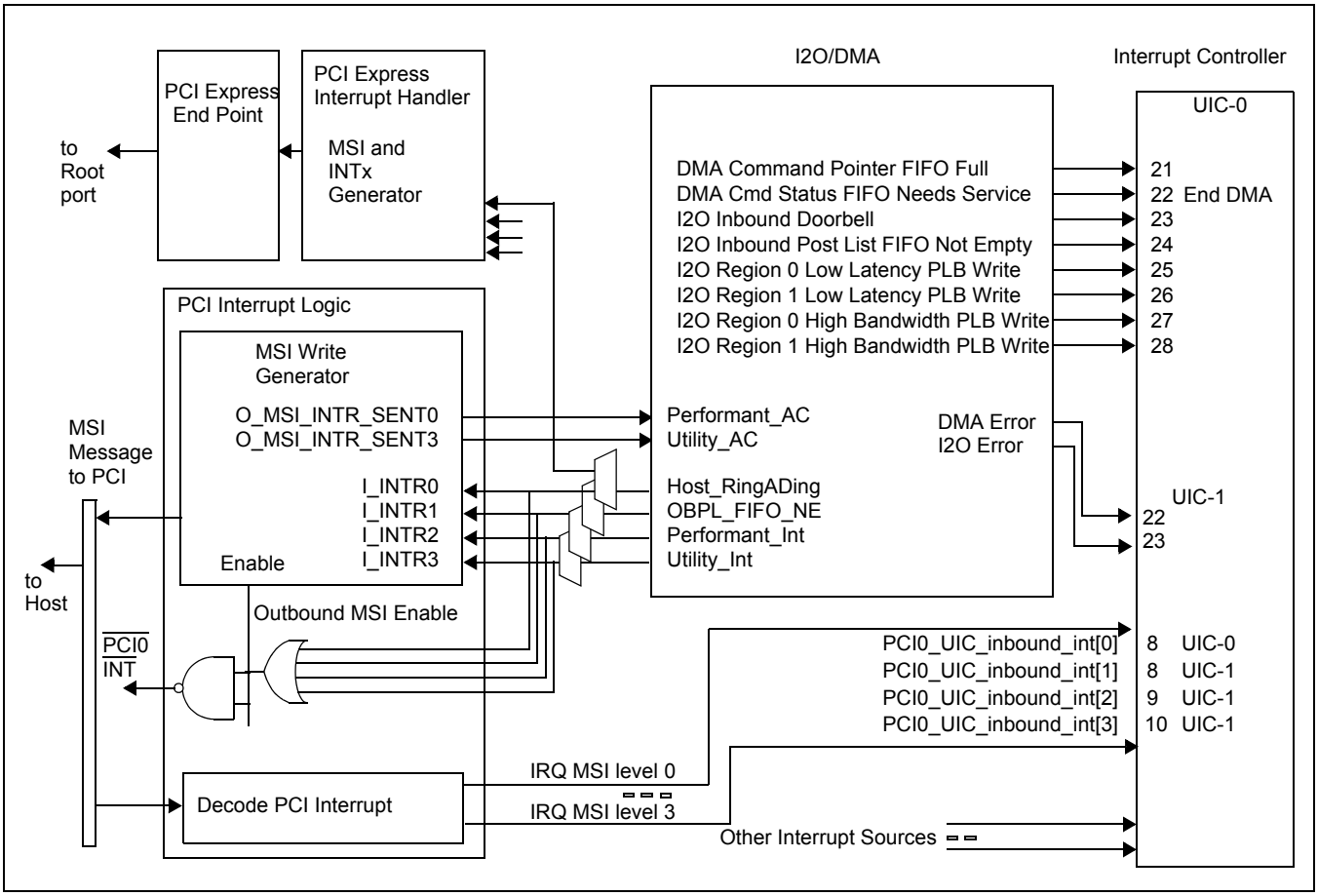
The interrupt controller mask must be enabled for the interrupt to be active. There is no way to "clear" the interrupt since it is asserted for a single clock.

Figure 26-3. PLB Write Interrupt Region Windows



This provides a service similar to "message signaled interrupts". Some adapters (Fiber Channel and SCSI) can be configured to write interrupt information to a memory address rather than assert an interrupt signal. This has the same advantages of PCI MSIs: the write flushes data so the 460EX/EXr/GT does not have to read the interrupting device buffers. The interrupt can be uniquely identified by reading memory, which is faster than reading the interrupting device.

Figure 26-4. I2O/DMA Interrupt Sources



26.2.6 DMA Register PLB Slave Module

This PLB Slave is shared between the I2O Register Interface and the two DMA Register Interfaces. This slave only accepts/responds to single-beat (1–16 byte) memory read and write cycles. The PLB slave contains a single request buffer for both read and write transfers. All transactions are completed in the order they are address acknowledged on the PLB. While the slave does not support multiple outstanding requests at the same time, it does support PLB pipelining. It is able to respond to secondary PLB_SAValid requests. The PLB Slave also translates 128-bit PLB bus data down to 64 bits for the I2O/DMA Register Interface.

26.2.7 Slave Address Decode

When enabled, the PLB Slave's address range is controlled by the I2O_Base_Low (DCR address 0x66) and the I2O_Base_High (DCR address 0x67) registers. This defines a 4KB-aligned region that may be located anywhere within PLB address space.

If enabled, the PLB Slave responds to single-beat memory requests whose PLB_Address[63:12] matches the programmed (I2O_Base_High, I2O_Base_Low) register.

26.2.8 Slave Request Buffer

This PLB Slave contains a single request queue. The request queue is shared by both read and writes to this slave. On writes, the request queue remains allocated until the write completes on the I2O or DMA register interface. On reads, the request queue remains allocated until the read completes on PLB. When the request queue is allocated, additional requests to this slave will be rearbitrated on PLB.

26.2.9 Error Handling

The PLB Slave can report incorrect byte alignment and aborted read errors by means of the DMA Status or I2O Status registers (an interrupt is generated).

On a detected error, the error status, master ID, master size, and byte enables are latched in the Slave Error Attribute Register and the transfer address is latched in the Slave Error Address Register. The slave always locks the Slave Error Attribute and Address registers on the first error detected.

The Slave Error Attribute and Slave Error Address registers are shared by I2O and DMA register interfaces. The Slave Error Attribute and Address registers are aliased such that both I2O and DMA register interfaces access the same Slave Error registers.

26.2.10 I2O/DMA Register Operation

During an I2O Register Interface read to any of the Queue Ports where its corresponding posting register is empty and the corresponding FIFO is empty, then a read results in 0x0 FFFF_FFFF value.

During a DMA Register Interface read to any of the Queue Ports where its corresponding posting register is empty and the corresponding FIFO is empty, then a read results in 0x0 0000_0000 value.

A read request to any of the Queue Ports in the I2O/DMA Register Interface will be retried/rearbitrated if the corresponding prefetch register is empty and the corresponding FIFO is not empty. Otherwise a deadlock situation will occur.

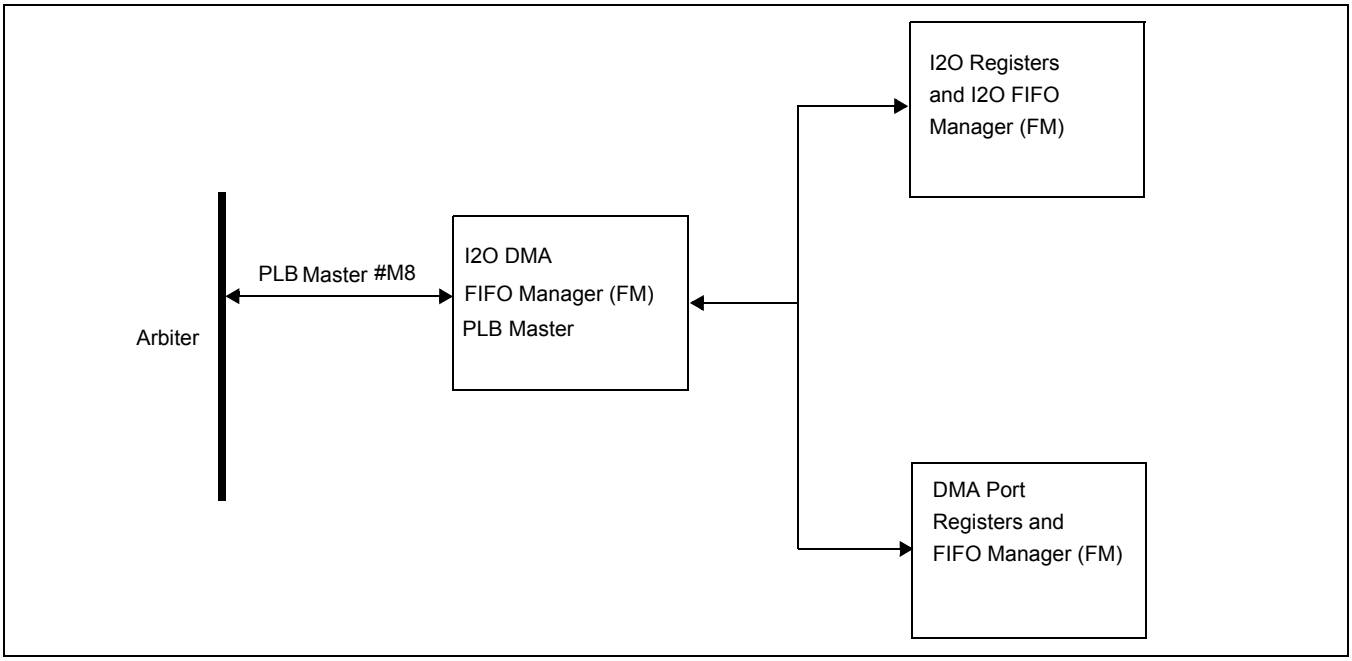
During an I2O/DMA Register Interface write to any of the Queue Ports where its corresponding posting register is full and the corresponding FIFO is not full, the write can be posted as long as all the data can be posted. If the write data cannot be posted, the cycle is rearbitrated.

During an I2O/DMA Register Interface write to any of the Queue Ports where its corresponding posting register is full and the corresponding FIFO is full, the transaction will be completed gracefully (all handshaking completes normally), with the write data being ignored. The appropriate interrupt status bit in the 460EX/EXr/GT Interrupt Status Register is set. If the interrupt is not masked, the 460EX/EXr/GT is interrupted. In addition, the PLB Slave will generate a PLB error to the requested master.

26.2.11 FIFO Manager (FM) PLB Master Module

The I2O and DMA FIFO Managers share a common PLB connection through the I2O/DMA FIFO Manager PLB Master #M8, as shown in the following figure.

Figure 26-5. I2O and DMA FIFO Managers



26.2.11.1 FM PLB Master Supported PLB Transfers

The I2O/DMA FIFO Manager PLB master performs 128-bit transfers. Transfers are always unlocked and single-beat.

26.2.11.2 FM PLB Master Error Handling

On the assertion of PLB_MWrErr, PLB_MRdErr, or a PLB bus size error for a given request, the PLB master informs the originator of the error and completes the current tenure.

The assertion of PLB_MIRQ is asynchronous to the transfer that caused the error. Therefore, the PLB master cannot associate the transfer request with the error. To minimize the possibility of data corruption, the DMA Engine can halt when PLB_MIRQ is asserted. Software must interrogate the PLB Slave's error address and error attribute registers to determine which transfer caused the error. If programmed this way, the error address register and error attribute register will be valid for only the first error encountered; the registers must be cleared before they can save any other error information.

26.2.11.3 FM PLB Master Transfer Attributes

These signals are driven by the master to present specific transfer information to slaves, which monitor these signals. These attributes are asserted by the master with the Mn_request signal and remain valid until the clock cycle following the assertion of PLB_MnAddrAck, PLB_MnRearbitrate.

Table 26-2. FM PLB Master Transfer Attributes Summary

Signal Name	Used by FM PLB Master	Expected Slave Behavior
Mn_guarded	Yes (tied active)	Access only what the master requested
Mn_ordered	Yes (tied active)	Do not allow PLB reads to pass PLB writes

26.2.12 DMA Data Transfer PLB Master Module (DTE PLB Master)

The following sections describe the I2O DMA operations.

26.2.12.1 DMA DTE PLB Master Supported PLB Transfers

The DMA Data Transfer Engine (DTE) PLB masters perform 128-bit transfers only. Transfers are always unlocked and are either single beat or fixed length burst transfers.

26.2.12.2 DMA DTE PLB Master Error Handling

On the assertion of PLB_MWrErr, PLB_MRdErr, or a PLB bus size error for a given request, the PLB master informs the originator of the error and completes the current tenure.

The assertion of PLB_MIRQ is asynchronous to the transfer that caused the error. Therefore, the PLB master cannot associate the transfer request with the error. To minimize the possibility of data corruption, the DMA Engine can halt when PLB_MIRQ is asserted. Software must interrogate the PLB Slave's error address and error attribute registers to determine which transfer caused the error. If the No Halt on Error bit is asserted ("Memory Mapped DMA Registers" on page 915), the DMA Engine will not halt when PLB_MIRQ is asserted. If programmed this way, the error address register and error attribute register will be valid for only the first error encountered; the registers must be cleared before they can save any other error information.

26.2.12.3 DMA DTE PLB Master Alignment

The DTE PLB master attempts to align to either 64- or 16-byte boundaries. This behavior is programmable through the DMA Configuration Register (see *DMA Configuration Register (I2O0_DMAx_CFG)* on page 920) with 16-byte alignment set as the default.

For 16-byte Alignment:

If the starting address is not 16-byte aligned, the DTE PLB Master will perform a single-beat transfer to become 16-byte aligned. Once aligned to a 16-byte boundary, transfers will be broken up into bursts. The DTE PLB Master proceeds to generate 512-byte bursts until the remaining byte count is less than 512 bytes. Once this happens, the Master performs one final burst to complete the transfer. If the transfer does not complete on a 16-byte boundary, then the Master performs one final single-beat transfer to complete the transfer.

For 64-byte Alignment:

The behavior in this case is nearly identical to 16-byte alignment with one exception. After the original single-beat transfer to become 16-byte aligned, the DTE PLB Master will perform a burst to become 64-byte aligned. Then the series of bursts (described for 16-byte alignment) begin and complete as they did for 16-byte alignment.

26.2.12.4 DMA DTE PLB Master Slave Terminated Transactions

If a PLB slave terminates a fixed-length burst transaction prematurely, the PLB master will continue to retry the transaction until completion. Completion status is not returned to the requestor until the transaction is complete.

26.2.12.5 DMA DTE PLB Master Write Byte Count

The DTE PLB master delivers, as a minimum, the number of bytes as indicated by Mn_BE and Mn_size PLB signals. A master always delivers the indicated byte count unless:

- The slave prematurely aborts/disconnects the transfer
- PLB_MWrErr is asserted
- Unsupported PLB width

26.2.12.6 DMA DTE PLB Master Read Byte Count

The DTE PLB master accepts, as a minimum, the number of bytes as indicated by Mn_BE and Mn_size PLB signals. A master always accepts the indicated byte count unless:

- The slave prematurely aborts/disconnects the transfer
- PLB_MRdErr is asserted
- Unsupported PLB width

26.2.12.7 DMA DTE PLB Master Transfer Attributes

These signals are driven by the master to present specific transfer information to slaves, which monitor these signals.

Table 26-3. DTE PLB Master Transfer Attributes Summary

Signal Name	Used by DTE PLB Master	Expected Slave Behavior
Mn_new_context	Yes	Flush prefetch buffers (Special Buffers)
Mn_guarded	Yes (set if Mn_prefetch_1k and Mn_prefetch_2k are both inactive)	Access only what the master requested
Mn_prefetch_1k	Yes	Prefetch 1K worth of data
Mn_prefetch_2k	No	Prefetch 2K worth of data
Mn_ordered	Yes	Do not allow PLB reads to pass PLB writes
Mn_relaxed_order	Yes	PCI relaxed ordering request
Mn_no_snoop	Yes	PCI no snoop request

The Mn_new_context signal is asserted when the DTE PLB Master is starting a new DMA operation. It directs the addressed slave to flush any prefetched data that it might have for this address range and to go obtain new data.

The Mn_guarded signal is only asserted when both Mn_prefetch_1k and Mn_prefetch_2k are not asserted. If this signal is asserted, it tells the slave to restrict access to exactly what the master requested.

The Mn_prefetch_1k signal tells the addressed slave that it should prefetch 1K of data starting at the address requested by the master.

Note: The DMA Configuration Register includes a "Prefetch Disable" bit ([DXEPD] - bit 31), which will prevent the assertion of Mn_prefetch_1k.

The Mn_prefetch_2k signal tells the addressed slave that it should prefetch 2K of data starting at the address requested by the master. This signal will not be asserted by the DMA Engine.

The Mn_ordered signal tells the slave to complete the associated write transfer before allowing any read from the same area.

The Mn_relaxed_order signal, when asserted, tells the PCI slave to allow relaxed ordering on the PCI bus.

The Mn_no_snoop signal, when asserted, tells the PCI slave to use no snoop for this data transfer on the PCI bus.

26.2.13 I2O Registers and I2O FIFO Manager Module

The I2O FIFO Manager is responsible for transferring Message Frame Addresses (MFAs) to/from the I2O Queues in memory in response to I2O Register reads/writes. The I2O FIFO Manager accomplishes this data movement by utilizing the FIFO Manager PLB Master.

For I2O "pull" operations, the I2O FIFO Manager is responsible for obtaining and completing a partially defined Free CDB to move the Message Frame from host memory to 460EX/EXr/GT system memory. Once all necessary fields of the Free CDB are filled out, the I2O FIFO Manager passes a pointer to the completed CDB (referred to as a Command Descriptor Block Pointer (CP)) to the DMA Register Interface. The I2O FIFO Manger places the CP in the DMA Command Descriptor Block Pointer Post List FIFO (CP FIFO) by means of the dedicated port between the I2O FIFO Manager and the DMA Register Interface.

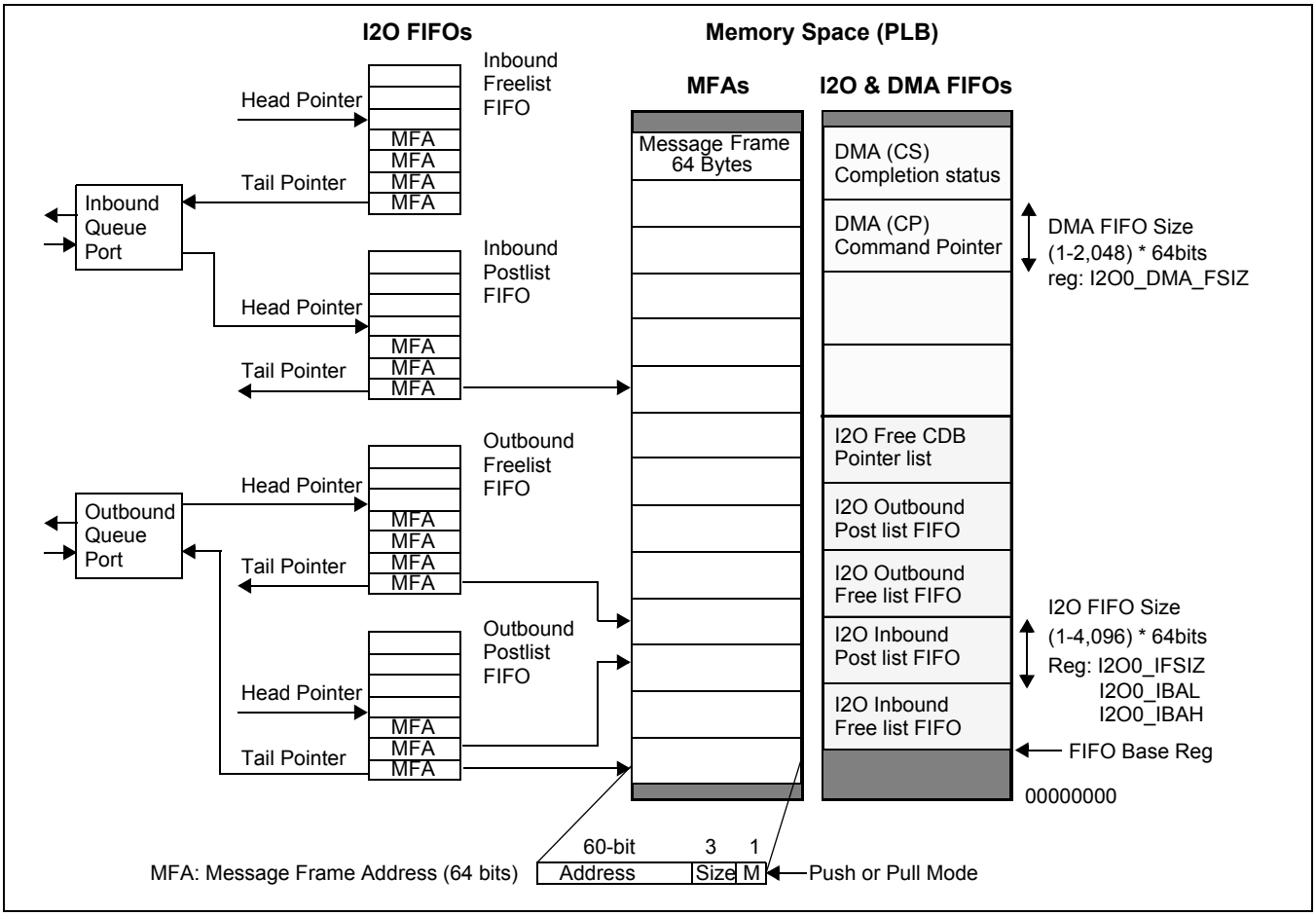
26.3 I2O Operation

The following sections describe the operation of the I2O function.

26.3.1 Overview

The I2O function provides a messaging service between a host processor and one or more IOPs, such as the PPC460EX/EXr/GT. Each locally attached IOP provides the default message transport mechanism where messages reside in a shared memory structure, the Message Frame (MF). The MF data structure is defined by the I2O specification. A Message Frame can exist anywhere in memory. The memory of a Message Frame is contiguous, but not all Message Frames are located together/contiguously. A typical MF is 64-bytes in size and starts on a 4-byte ("push" mode) or 16-byte ("pull" mode) address boundary. An MF is referred to by its Message Frame Address (MFA). The MFA specifies the first byte of the message frame header.

Figure 26-6. I2O Messaging Queues



MFAs are managed with queues: inbound and outbound message queues. The inbound message queue is for messages traveling from a source that does not implement the queues (usually the host) to the destination that implements the queues (usually the 460EX/EXr/GT). The inbound message queue contains MFAs that point to messages in the 460EX/EXr/GT system memory space.

The outbound message queues are for messages traveling from a source that implements the queues (usually the 460EX/EXr/GT) to the destination that does not implement the queues (usually the host). The outbound message queue contains MFAs that point to messages in host memory space.

Each of these queues holds MFAs for both messages that require processing (that is, post list FIFO), and of message frames that have been processed (that is, a free list FIFO). It is the responsibility of the I2O logic to manage the queues.

The queue status is communicated to the host and the 460EX/EXr/GT in a number of different ways. The host is informed the Outbound Post List FIFO is not empty through a host interrupt. The 460EX/EXr/GT is informed the Inbound Post List FIFO is not empty through a 460EX/EXr/GT interrupt. A FIFO is considered empty if a read from the FIFO returns all Fs. It is the responsibility of the message producer to ensure that the FIFO is not full as a write to a full FIFO results in the write being ignored (resulting in a 460EX/EXr/GT interrupt).

The size of the FIFOs vary from implementation to implementation. The minimum FIFO size is 1 MFA, and the maximum supported by this design is 4096 MFAs.

Access to the MFA queues is accomplished through the I2O Register Interface of the IOP. The I2O Register Interface maps into both the host memory map and the 460EX/EXr/GT system memory map (i.e. shared). The I2O Register Interface logic consists of the control logic needed to manage the FIFOs (head pointer, tail pointer, base address, etc.).

An MFA is placed into a FIFO by writing to the I2O Register Interface; an MFA is removed from a FIFO by reading from the I2O Register Interface.

Table 26-4. I2O Register Interface Queue Access Ports

Register	Mnemonic	Function	Offset
Host Inbound Queue Port	I2O0_IHIQ	A read of this register returns an MFA from the Inbound Queue Free List.	40
		A write to this register places an MFA in the Inbound Queue Post List.	
460EX/EXr/GT Inbound Queue Port	I2O0_IOPIQ	A read of this register returns an MFA from the Inbound Queue Post List.	50
		A write to this register places an MFA in the Inbound Queue Free List.	
Host Outbound Queue Port	I2O0_IHOQ	A read of this register returns an MFA from the Outbound Queue Post List.	44
		A write to this register places an MFA in the Outbound Queue Free List.	
460EX/EXr/GT Outbound Queue Port	I2O0_IOPOQ	A read of this register returns an MFA from the Outbound Queue Free List.	5C
		A write to this register places an MFA in the Outbound Queue Post List.	

Accessing the I2O registers causes the I2O controller to perform memory Reads or Writes depending on the type and address of the I2O Register Interface access. The data storage elements for the queues/FIFOs are usually implemented within large memory space. The FIFOs starting address on PLB is indicated by the FIFO Base Address Register. Each individual FIFO is an offset from the FIFO Base Address as indicated in the table below.

Table 26-5. I2O FIFO Address/Offsets

FIFO Name	Address Range	
I2O Inbound Free List	Start =	FIFO Base Register
	Length =	(I2O FIFO Size Register + 1) * 8
	End =	Start + Length – 1
Values above are valid only if the Inbound Free List FIFO is enabled (Length=0 otherwise)		
I2O Inbound Post List	Start =	I2O Inbound Free List End + 1
	Length =	(I2O FIFO Size Register + 1) * 8
	End =	Start + Length – 1
The values shown here are valid only if the Inbound Post List FIFO is enabled (Length=0 otherwise)		
I2O Outbound Free List	Start =	I2O Inbound Post List End + 1
	Length =	(I2O FIFO Size Register + 1) * 8
	End =	Start + Length – 1
The values shown here are valid only if the Outbound Free List FIFO is enabled (Length=0 otherwise)		
I2O Outbound Post List	Start =	I2O Outbound Free List End + 1
	Length =	(I2O FIFO Size Register + 1) * 8
	End =	Start + Length – 1
Values above are valid only if the Outbound Post List FIFO is enabled (Length=0 otherwise)		
I2O Free CDB Pointer List	Start =	I2O Outbound Post List End + 1
	Length =	(I2O FIFO Size Register + 1) * 8
	End =	Start + Length – 1
The values shown here are valid only if the I2O Free CDB Pointer FIFO is enabled (Length=0 otherwise)		

26.3.2 Inbound and Outbound MFAs

An MFA (Message Frame Address) is a 64-bit entity that represents the starting address of the Message Frame. Inbound MFAs allow only the most significant 60 bits for use as physical address. The remaining 4 bits are used to specify the size of the Message Frame and whether to use I2O pull or I2O push mode.

Table 26-6. Inbound Post List MFA Encoding

64-bit MFA Encoding				
63 MSB	4	3	1	0
Address		Message Size index		Mode

Note: A Mode field value of 1 indicates I2O pull mode. A Mode field value of 0 indicates I2O push mode.

I2O pull mode requires that the Message Frame reside on a 32-byte boundary. Thus, the least significant bit of the 60 address bits must be zero for I2O pull operations. The Message Size field for an MFA corresponds to one of eight I2O0_MFACx Count registers. Each I2O0_MFACx Count register indicates a message size in multiples of 16 bytes.

Example:
- Message Size index field = 0b101, then the I2O0_MFAC5 (Count 5) register would be used to determine the message size.
- I2O0_SGC5 (Count 5) = 0b01, then the message size would be 2 * 16 = 32 bytes.

For I2O push mode, the message size bits are reserved.

For Inbound Post List MFs located in an 460EX/EXr/GT's shared memory, the address contained in the MFA is the offset between the start of the target 460EX/EXr/GT's Register Interface (PCI I2O BAR) and the start of the message. For MFs located in host memory, the address contained in the MFA is the physical address of the message.

For Outbound MFAs, all 64 bits are used as the physical address of the Message Frame.

26.3.3 FIFO Prefetching and Posting

To improve latency, the I2O logic both posts and prefetches MFAs. An MFA write to a FIFO is posted in their individual posting registers before being delivered to the FIFOs (by means of the PLB). Additional writes to a FIFO while its posting registers are full are retried until the data can be accepted.

Read data from each FIFO is also automatically prefetched and held in a staging register. The automatic read is initiated when the FIFO is not empty and the staging register is empty. If a FIFO is read and the FIFO is not empty, but the staging register is empty, the read access is retried.

26.3.4 I2O Error Handling

The following error conditions can be reported by I2O:

- Incorrect byte alignment (PLB Slave)
- A write to a full FIFO (PLB Slave)
- A read or write to a FIFO that has not been enabled (PLB Slave)
- PLB_MRdErr or PLB_MWrErr (PLB Master)
- Wrong PLB Slave Size (PLB Master)

Errors accessing the DMA registers through the PLB Slave or by the I2O FIFO Manager are reported in the DMA Status Register's PLB Slave Error bits. In addition, the PLB Slave Error Attribute and PLB Slave Error Address registers can be queried for further diagnostics.

An error reported by the DMA FIFO Manager PLB Master is reported in the DMA Status Register's PLB Error bits.

An error that occurs while the I2O FIFO Manager is accessing the PLB is indicated by the PLB_MRdErr or PLB_MWrErr signals. Assertion of these error signals is latched in the I2O Status Register (resulting in and interrupt).

Any error reported by the I2O FIFO Manager PLB Master will change the behavior of the I2O Queue ports until the error condition is cleared. Further writes to a Queue Port will be absorbed with normal status on PLB. However, the data will be ignored and the FIFO will not be updated. Further reads from a Queue Port will return all Fs (an empty FIFO indicator) with normal completion on the PLB. All writes and prefetches to and from the I2O FIFOs will be suspended.

Writing 1s to the appropriate I2O Status Register bits can clear errors and resume normal operation.

26.3.5 Interrupt Delay Modes

The I2O logic includes interrupt delay logic. The logic attempts to collect several Host bound interrupts so that the interrupt handle can take care of all of them at one time. This logic will work in two different modes, leading edge, and trailing edge.

In leading edge mode, when any interrupt is detected as going active, the interrupt delay counter is started. The counter counts down from the Interrupt Delay Value. While the counter is non-zero, all interrupts are masked. When the counter reaches 0, the interrupts are unmasked. When all of the interrupts are serviced, the counter is reloaded and waits for another interrupt.

In trailing edge mode, when all interrupts have been detected as being in active, the interrupt delay counter is started. The counter counts down from the Interrupt Delay Value. While the counter is non-zero, all interrupts are masked. When the counter reaches 0, the interrupts are unmasked. When all the interrupts have been serviced, the counter is reloaded and begins to count down.

The Interrupt Delay logic is controlled by a 32-bit register in I2O memory mapped register space (register offset = 0x98). The most significant bit (bit 31) will be used to control which technique to use (1=leading edge, 0=trailing edge). Bits 23-0 will be the load value for the counter. To disable the logic completely, a load value of 000h must be used. The logic automatically stops blocking the interrupt because the counter always remains at 000h, which is the non-blocking value.

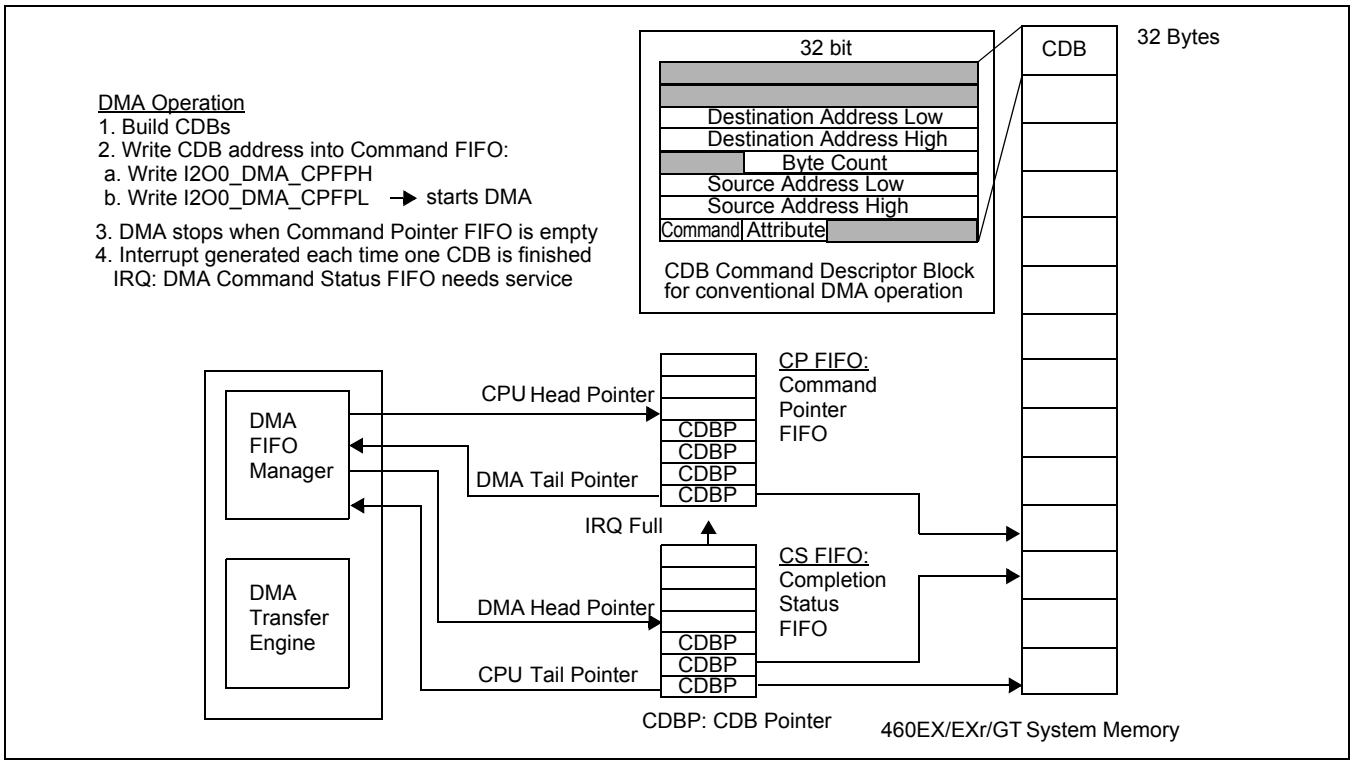
26.4 DMA Operation

The following sections cover DMA operation as it relates to I2O.

26.4.1 Overview

The DMA FIFO Manager and DMA Data Transfer Engine pair is responsible for moving data from a source location to a destination location. The data can be either I2O message Frames or application data. The DMA Data Transfer Engine can transfer data to/from any source and destination. The DMA Data Transfer Engine also performs memory-fill and memory-check commands.

Figure 26-7. DMA Queues



The DMA Data Transfer Engine operations/commands are communicated to it through a data structure known as a Command Descriptor Block (CDB). The CDB is constructed by the 460EX/EXr/GT and is placed in 460EX/EXr/GT system memory. The CDB is referenced by a 64-bit Command Descriptor Block Pointer (also referred to as Command Pointer or simply CP). The CP is what the DMA Data Transfer Engine uses to access the CDB. The CDB pointers are contained in two FIFOs and are managed by the DMA FIFO Manager. The first FIFO is a Command Pointer FIFO, and the second is a Completions Status FIFO (also referred to as the CS FIFO).

Each CP FIFO entry is a pointer to a CDB that needs to be processed. The DMA Data Transfer Engine continues to process CDBs any time the CP FIFO is not empty. If the CP FIFO is not empty, then the DMA Data Transfer Engine fetches the CDB contents for the location specified by the CP and executes the command. When the DMA Data Transfer Engine completes the requested operation, it adds status to the CP and places the CP in the Completed Command Descriptor Block Pointer with Status FIFO (also referred to as the Completion Status FIFO or simply CS FIFO). The 460EX/EXr/GT is interrupted when the CS FIFO is full or when an interrupting CP is placed in the CS FIFO.

The size of each FIFO is programmable. The maximum FIFO sizes are 2048 64-bit CPs. 32-bit CPs are supported, but they are internally stored as 64-bit CPs in the FIFO. Each FIFO is contained in contiguous memory and *Table 26-7* gives their starting memory addresses.

Table 26-7. DMA FIFO Address/Offset

FIFO Name	Address Range	
DMA Command Pointer (CP)	CP Start =	I2O Free CDB Pointer FIFO's End + 1
	CP Length =	(DMA FIFO Size Register + 1) * 8
	CP End =	CP Start + CP Length – 1
DMA Completion Status (CS)	CS Start =	CP End + 1
	CS Length =	(DMA FIFO Size Register + 1) * 8
	CS End =	CS Start + CS Length – 1
Values are valid only if DMA FIFOs are enabled.		

The 460EX/EXr/GT updates the FIFOs through the DMA register Interface. The DMA Register Interface is memory mapped in the PLB address space. The DMA FIFO Manager consists of the control logic needed to manage the FIFOs (head pointers, tail pointers, etc.). The data storage elements for the queues/FIFOs are usually implemented with one large memory space. Accessing the DMA registers causes the DMA FIFO Manager to perform memory reads or writes depending on the type and address of the DMA Register Interface access.

A CP is placed in the CP FIFO by writing to the DMA Register Interface and a CS is removed from the CS FIFO by reading from the DMA Register Interface.

Table 26-8. DMA Register Interface Queue/FIFO Access Ports

Register	Mnemonic	Function
DMA Command Pointer FIFO Port	CP_PORT	A read of this register returns 0.
		Writes to this register places a CDB Pointer in the Command Pointer FIFO.
DMA Completion Status FIFO Port	CS_PORT	A read of this register returns a CDB Pointer with status from the Completion Status FIFO.
		Writes to this register are ignored.

26.4.2 DMA Register Interface and FIFO Manager

The DMA FIFO Manager is responsible for managing the CP and CS FIFOs. In response to DMA register accesses, the DMA FIFO Manager places CPs in memory and updates the FIFO pointers.

The CP is either a 64-bit or 32-bit entity with the address size encoded in bit 2 of the CP. Setting bit 3 disables normal interrupt-generation when the CP is placed in the CS FIFO. The least significant 4 bits of a CP are not address bits, forcing CDBs to be 16-byte aligned.

Table 26-9. DMA CDB Pointer Encoding

Bit	Field Name	Description
63:32	CDB Pointer	Upper 32 bits of a Command Descriptor Block address
31:4	CDB Pointer	Bits 31:4 of a Command Descriptor Block address.
3	No Int	**Set**: Do not generate an interrupt when CDB processing is complete. **Clear**: Generate an interrupt when CDB processing is complete.
2	Address Size	**Set**: This is a 64-bit Command Descriptor Block address. **Clear**: This is a 32-bit Command Descriptor Block address.
1:0	Reserved	Reserved

The CS is either a 64-bit or 32-bit entity with the address size encoded in bit 2. Bit 3 signifies whether this CS generated an interrupt when placed in the CS FIFO. If a CS entry signals an interrupt, the interrupt will remain active until all interrupt-capable CS entries are removed from the CS FIFO.

Table 26-10. DMA CDB Pointer Status Encoding

Bit	Field Name	Description
63:32	CDB Pointer	Upper 32 bits of a Command Descriptor Block address
31:4	CDB Pointer	Bits 31:4 of a Command Descriptor Block address.
3	No Int	**Set**: This CDB did not generate an interrupt when CDB processing was completed. **Clear**: This CDB generated an interrupt when CDB processing was completed.
2	Address Size	**Set**: This is a 64-bit Command Descriptor Block address. **Clear**: This is a 32-bit Command Descriptor Block address.
1:0	Status	These bits are the status returned by the DMA Data Transfer Engine for a completed Command Pointer. Encoding is as follows: 00 DMA operation completed normally 01 Data Check returned with a miscompare. 10 Error fetching from Command Status FIFO (address size and field are invalid) 11 CDB encountered an error, but did not halt the engine.

26.4.3 FIFO Prefetching and Posting

To improve latency, the DMA FIFO Manager both posts and prefetches CPs and CSs. A CP or CS write to a FIFO is posted in their individual posting registers before being delivered to the FIFOs. Additional writes to a FIFO while its posting registers are full are retried until the data can be accepted.

Read data from each FIFO is also automatically prefetched and held in a staging register. The automatic read is initiated when the FIFO is not empty and the staging register is empty. If a FIFO is read and the FIFO is not empty but the staging register is empty, the read access is retried.

26.4.4 DMA FIFO Manager Error Handling

The DMA Register Interface and FIFO Manager can report the following error conditions:

- Select errors that include unsupported byte alignment. (PLB Slave)
- A write to a full FIFO. (PLB Slave)
- A read or write to a DMA FIFO that is not enabled. (PLB Slave or I2O)
- High-Low port access order. (PLB Slave)
- PLB_MRdErr or PLB_MWrErr (PLB_Master)
- Wrong PLB Slave Size (PLB Master)

Errors accessing the DMA registers through the PLB Slave or by the I2O FIFO Manager are reported in the PLB Slave Error bits of the DMA Status Register. An error reported by the DMA FIFO Manager PLB Master is reported in the PLB Error bits of the DMA Status Register.

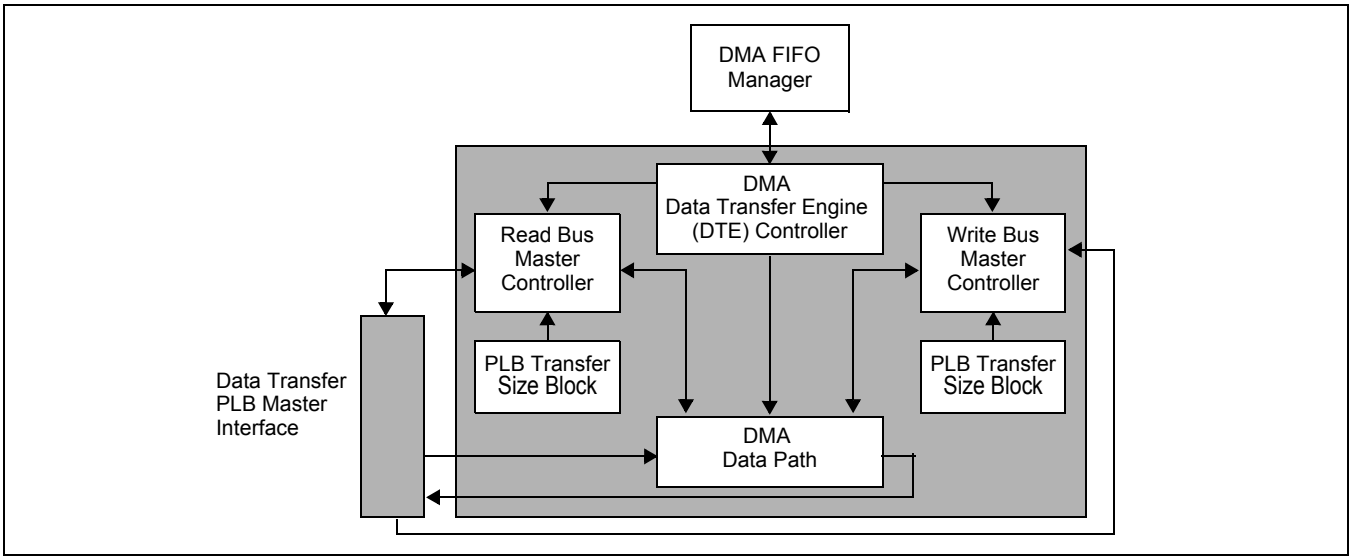
On errors accessing the DMA registers through the PLB Slave, the slave will assert Sl_MRdErr for read errors, but the slave will not assert Sl_MWrErr on writes. In either case, the error type is latched in the DMA Status Register, resulting in an interrupt. In addition, the PLB Slave will also latch the transfer attributes: address, byte enables, read/write, master ID, and master size.

Errors that occur while the DMA FIFO Manager is accessing the PLB bus are indicated by the PLB_MRdErr and PLB_MWrErr signals. PLB error assertion is latched in the DMA Status Register (resulting in an interrupt). Any error reported by the FIFO Manager's PLB master will change the behavior of the CP and CS ports until the error condition is cleared. Further writes to the CP port will be absorbed with normal status on the PLB Slave. However, the data will be thrown away and the FIFO will not be updated. Further reads from the CS port will return all zeroes with normal completion on the PLB. All writes and prefetches to/from the DMA FIFOs will be suspended.

26.4.5 DMA Data Transfer Engine (DTE)

The DMA Data Transfer Engine is responsible for interpreting CDBs and performing the requested memory transfers. *Figure 26-8* shows the basic block diagram of the DMA Data Transfer Engine.

Figure 26-8. DMA Data Transfer Engine Block Diagram



26.4.5.1 DMA Data Transfer Engine Controller

This module is responsible for DMA transaction scheduling, master participation selection, and DMA status updates.

26.4.5.2 Bus Master Controllers

These modules are responsible for scheduling appropriate sized requests to the Bus Master (PLB) so that the DMA Data Path queues never overflow or underflow. This includes performing the multicast function. The Bus Master Controllers shown above are bus independent. Bus personality is achieved through the PLB Transfer Size block.

26.4.5.3 PLB Transfer Size

This module is responsible for calculating the next request size to be made by the corresponding PLB Bus Master Controller. This effectively breaks up large DMA requests into sizes/alignments compatible with the appropriate bus and DMA Data Path queues.

26.4.5.4 Data Transfer PLB Bus Master

This module performs PLB bus specific protocols to fulfill the PLB Bus Master Controller's data transfer request.

26.4.5.5 DMA Data Path

This module provides three functions. The first function is data buffering, allowing the bus masters to operate at different effective frequencies, to do the multi-cast function, and to do single master moves. The second function is byte alignment (allowing any byte to be transferred to any address). The third function is memory pattern fill/check functionality.

26.4.5.6 DMA Data Transfer Engine Error Handling

On a fatal error, the DMA Data Transfer Engine stops the current operation and the DMA Status Register reports the cause of the error, resulting in an interrupt. Errors that can cause the DMA Data Transfer Engine to stop include:

- PLB master PLB_MWrErr, PLB_MRdErr, or PLB_MIRQ assertion
- Illegal opcode
- DMA FIFO Manager's PLB master PLB_MRdErr while reading the CP FIFO

The DMA engine will halt as described below:

When halted, the DMA Data Transfer Engine goes to and remains in a HALT state until the DMA Halted bit is cleared in the DMA Status register. A write of a 1 to the DMA Status register Halted bit restarts the DMA Engine and causes it to fetch a new CP. The command that caused the error is not returned to the CS FIFO; it is discarded.

Interrupt generation for Halted on Error is handled by the IOP Interrupt Status Register in the I2O module. Any error reported in the DMA Status register will set the DMA Error bit in the IOP Interrupt Status register, but may only be cleared from the DMA Status register.

If PLB_MIRQ is asserted while the DMA Engine has a transfer in progress, the engine associated with that PLB_MIRQ is halted. The DMA Engine PLB Master Status bits and PLB_MIRQ bit of the DMA Status registers for the halted DMA port will reflect the error condition. If PLB_MIRQ is asserted while the DMA Engine does not have a transfer in progress, the engine will be halted to prevent a transfer from being initiated and only the PLB_MIRQ bit of the DMA Status registers will be set.

26.5 DMA Commands

This section describes the DMA commands and related components.

26.5.1 Basic Command Descriptor Block Structure

The information for DMA operations (such as Source and Destination addresses and Byte count) is defined in DMA Command Descriptor Blocks (CDBs) in main memory. The following table shows the generic DMA Command Descriptor Block structure.

Table 26-11. Basic CDB (Command Descriptor Blocks) Structure

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Lower Half of SG3 Address			
0x18	Upper Half of SG3 Address			
0x14	Lower Half of SG2 Address			
0x10	Upper Half of SG2 Address			
0x0C	Reserved	SG (Scatter/Gather) Count		
0x08	Lower Half of SG1 Address			
0x04	Upper Half of SG1 Address			
0x00	DMA OpCode	Attributes	Reserved	

Note: A single CDB structure occupies 32 bytes and is located on a 16-byte boundary. The DMA Data Transfer Engine always fetches 32 bytes when accessing a CDB. The DMA engine only supports the simple addressing Scatter/Gather List (SGL) format defined in I2O (v1.5). No other SGL formats are supported.

Each **SG Address** field is used to specify the beginning memory location for an SGL. This field must be 64 bits wide. Each SGL referenced must be entirely contained within memory and must not overlap. Otherwise, unexpected results (such as memory thrashing) will occur.

The **SG Count** field specifies the total byte count to transfer during the given operation. Data will be written to or read from the locations defined by the SGL(s) until the byte count is satisfied. A Scatter/Gather (SG) Count value of 0 is equivalent to a NOP operation. The maximum SG Count value allowed is 16,777,215 bytes (16MB – 1) = 0xFF_FFFF.

The **DMA Opcode** field tells the DMA engine what function to perform. Use only the opcodes shown in the following table. All others are reserved. Using a reserved Opcode will cause the DMA engine to HALT and return an ILLEGAL OPCODE Error.

The **Attributes** field contains miscellaneous transfer settings and is described in the following table.

Table 26-12. Attributes Field Bit Definitions

Bits							
7	6	5	4	3	2	1	0
RlxOrder	NoSnoop	No M_Busy Wait	Reserved				

The RlxOrder bit causes the PCI relaxed ordering attribute to be asserted for all read/write operations required to complete the associated CDB.

The NoSnoop bit causes the PCI no snoop attribute to be asserted for all read/write operations required for the associated CDB.

The No M_Busy Wait bit specifies whether the DMA Data Transfer PLB Master should pause between PLB transfers for the targeted slave to deassert its M_Busy signal before proceeding with another transfer. This bit is significant only when dealing with PLB write transactions. When set, the DMA Data Transfer PLB Master will not wait for M_Busy to be deasserted. In that case, another transfer is submitted on PLB, or (if that was the last transaction required to complete a command) the Completion Status FIFO is updated to reflect the completed command. When using this setting, software must realize that the Completion Status entry may be retrieved before the targeted slave has finished handling the transfer internally. When clear, the DMA Data Transfer PLB Master will wait for M_Busy to be deasserted for every transaction required to complete the requested command.

The following table lists all valid DMA Opcodes:

Table 26-13. DMA Opcodes

Operation Mnemonic	Opcode								
	Binary								Hex
	7	6	5	4	3	2	1	0	
NO-OP	0	0	0	0	0	0	0	0	0x00
MOVE_SG1_SG2	0	0	0	0	0	0	0	1	0x01
MULTICAST	0	0	0	0	0	1	0	1	0x05
DATA_CHECK_128	0	0	1	0	0	0	1	1	0x23
DATA_FILL_128	0	0	1	0	0	1	0	0	0x24
LFSR_RESET	1	0	0	0	0	0	0	0	0x80
LFSR_CHECK	1	0	0	0	0	0	1	1	0x83
LFSR_FILL	1	0	0	0	0	1	0	0	0x84
LFSR_CHECK_INVERT	1	1	0	0	0	0	1	1	0xC3
LFSR_FILL_INVERT	1	1	0	0	0	1	0	0	0xC4

26.5.2 No Op

The No Op command causes the DMA Data Transfer Engine to return "no error" status and return to idle. The CP and status bits (always set to 0b00) are placed into the DMA CS FIFO when the operation is complete. *This command is intended for testing purposes only.*

Table 26-14. No Op CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Reserved			
0x18	Reserved			
0x14	Reserved			
0x10	Reserved			
0x0C	Reserved			
0x08	Reserved			
0x04	Reserved			
0x00	0x00	Attributes	Reserved	

26.5.3 MOVE_SG1_SG2

This command causes the DMA Data Transfer Engine to move the contents of the source buffer (SG1) to the destination buffer (SG2). The CP and status bits (always set to 0b00) are placed in the DMS CS FIFO when the operation is complete.

Table 26-15. MOVE_SG1_SG2 CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Reserved			
0x18	Reserved			
0x14	Destination Address Low (SG2)			
0x10	Destination Address High (SG2)			
0x0C	Reserved	SG Count		
0x08	Source Address Low (SG1)			
0x04	Source Address High (SG1)			
0x00	0x01	Attributes	Reserved	

26.5.4 MULTICAST

This command causes the DMA Data Transfer Engine to move the contents of the source buffer (SG1) to two distinct destination buffers (SG2 and SG3). The CP and status bits (always set to 0b00) are placed in the DMA CS FIFO when the operation is complete. Source and destination buffer addresses can start on any byte boundary, but the two destination buffers must have the same byte offset (SG2[3:0] = SG3[3:0]).

Table 26-16. MULTICAST CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Second Destination Address Low (SG3)			
0x18	Second Destination Address High (SG3)			
0x14	First Destination Address Low (SG2)			
0x10	First Destination Address High (SG2)			
0x0C	Reserved	SG Count		
0x08	Source Address Low (SG1)			
0x04	Source Address High (SG1)			
0x00	0x05	Attributes	Reserved	

26.5.5 DATA_CHECK_128

This command verifies that each 128-bit quadword (QWORD) in the source buffer (SG1) is identical to the specified 128-bit data pattern. The CP and status bits are placed in the DMA CS FIFO when the operation is complete. The status bits are set to 0b00 if each QWORD in the buffer matches the data pattern. The status bits are set to 0b01 if any mismatch is found.

The source buffer address must be a multiple of 16 bytes (SG1[3:0] = 0b0000), and the SG Count must also be a multiple of 16 bytes.

Table 26-17. DATA_CHECK_128 CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Lower DWORD of Data Pattern (SG3)			
0x18	Lower-middle DWORD of Data Pattern (SG3)			
0x14	Upper-middle DWORD of Data Pattern (SG2)			
0x10	Upper DWORD of Data Pattern (SG2)			
0x0C	Reserved	SG Count		
0x08	Lower Half of Region to Check (SG1)			
0x04	Upper Half of Region to Check (SG1)			
0x00	0x23	Attributes	Reserved	

26.5.6 DATA_FILL_128

This command fills the destination buffer with a 128-bit data pattern. The CP and status bits (always set to 0b00) are placed into the DMA CS FIFO when the operation is complete.

The destination buffer address must be a multiple of 16 bytes (SG2[3:0] = 0b0000), and the SG count must also be a multiple of 16 bytes.

Table 26-18. DATA_FILL_128 CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Lower DWORD of Data Pattern (SG3)			
0x18	Lower-middle DWORD of Data Pattern (SG3)			
0x14	Lower Half of Region to Fill (SG2)			
0x10	Upper Half of Region to Fill (SG2)			
0x0C	Reserved	SG Count		
0x08	Upper-middle DWORD of Data Pattern (SG1)			
0x04	Upper DWORD of Data Pattern (SG1)			
0x00	0x24	Attributes	Reserved	

26.5.7 LFSR_RESET

This command causes the current LFSR to be replaced by the seed value provided in the CDB. The CP and status bits (always set to 0b00) are placed into the DMA CS FIFO when the operation is complete.

Table 26-19. LFSR_RESET CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Reserved			
0x18	Reserved			
0x14	Lower DWORD of LFSR Seed Value (SG2)			
0x10	Lower-middle DWORD of LFSR Seed Value (SG2)			
0x0C	Reserved			
0x08	Upper-middle DWORD of LFSR Seed Value (SG1)			
0x04	Upper DWORD of LFSR Seed Value (SG1)			
0x00	0x80	Attributes	Reserved	

26.5.8 LFSR_CHECK

This command checks the contents of the specified buffer against the current LFSR value. The LFSR value changes with each data cycle. The CP and status bits are placed into the DMA CS FIFO when the operation is complete. The status bits are set to 0b00 if all the bytes in the buffer compare correctly. The status bits are set to 0b01 if any mismatch is found. Typically, this command is used in conjunction with the LFSR Fill command.

Table 26-20. LFSR_CHECK CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Reserved			
0x18	Reserved			
0x14	Reserved			
0x10	Reserved			
0x0C	Reserved	SG Count		
0x08	Lower Half of Buffer Address (SG1)			
0x04	Upper Half of Buffer Address (SG1)			
0x00	0x83	Attributes	Reserved	

26.5.9 LFSR_FILL

This command causes the specified buffer to be filled with values from the LFSR. The LFSR value changes with every data cycle. The CP and status bits (always set to 0b00) are placed into the DMA CS FIFO when the operation is complete. Typically, this command is used in conjunction with the LFSR Check command.

Table 26-21. LFSR_FILL CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Reserved			
0x18	Reserved			
0x14	Lower Half of Destination Buffer Address (SG2)			
0x10	Upper Half of Destination Buffer Address (SG2)			
0x0C	Reserved	SG Count		
0x08	Reserved			
0x04	Reserved			
0x00	0x84	Attributes	Reserved	

26.5.10 LFSR_CHECK_INVERT

This command checks the contents of the specified buffer against the current, *inverted* LFSR value. The LFSR value changes with each data cycle. The CP and status bits are placed into the DMA CS FIFO when the operation is complete. The status bits are set to 0b00 if all the bytes in the buffer compare correctly. The status bits are set to 0b01 if any discrepancy is found.

Table 26-22. LFSR_CHECK_INVERT CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Reserved			
0x18	Reserved			
0x14	Reserved			
0x10	Reserved			
0x0C	Reserved	SG Count		
0x08	Lower Half of Source Buffer Address (SG1)			
0x04	Upper Half of Source Buffer Address (SG1)			
0x00	0xC3	Attributes	Reserved	

26.5.11 LFSR_FILL_INVERT

This command causes the specified buffer to be filled with the *inverted* value from the LFSR. The LFSR value changes with each data cycle. The CP and status bits (always set to 0b00) are placed in the DMA CS FIFO when the operation is complete. (Typically this command is used in conjunction with the LFSR Check Invert command.)

Table 26-23. LFSR Fill Invert CDB

Offset	Byte 3	Byte 2	Byte 1	Byte 0
0x1C	Reserved			
0x18	Reserved			
0x14	Lower Half of Destination Buffer Address (SG2)			
0x10	Upper Half of Destination Buffer Address (SG2)			
0x0C	Reserved	SG Count		
0x08	Reserved			
0x04	Reserved			
0x00	0xC4	Attributes	Reserved	

26.6 I2O/DMA Registers

The I2O/DMA function is implemented with a set of DCR registers and a set of memory mapped registers as described below:

1. The registers described at *I2O/DMA Device Control Registers* are named with a prefix of I2O0_
2. *Memory Mapped DMA Registers* on page 915 describes the memory mapped registers. This section contains a detailed description for the registers. Register names use the I2O0_DMAx prefix.

26.6.1 I2O/DMA Device Control Registers

A read operation on reserved registers/bits always returns 0; a write operation on reserved registers results in an acknowledgment of the write, but the discarding of the write data.

Table 26-24. I2O/DMA Device Control Register Summary

Mnemonic	Register	Address	Access	Page
I2O0_LLIRQ0L	Low Latency PLB Write Interrupt Region 0 Base Low	0x0060	R/W	909
I2O0_LLIRQ0H	Low Latency PLB Write Interrupt Region 0 Base High	0x0061	R/W	909
I2O0_LLWR0M	Low Latency PLB Write Interrupt Region 0 Mask/Attribute	0x0062	R/W	909
I2O0_LLIRQ1L	Low Latency PLB Write Interrupt Region 1 Base Low	0x0063	R/W	910
I2O0_LLIRQ1H	Low Latency PLB Write Interrupt Region 1 Base High	0x0064	R/W	910
I2O0_LLWR1M	Low Latency PLB Write Interrupt Region 1 Mask/Attribute	0x0065	R/W	910
I2O0_IBAL	I2O Base Address Low	0x0066	R/W	911
I2O0_IBAH	I2O Base Address High	0x0067	R/W	911
	Reserved	0x0068		
I2O0_DMA_ST	DMA Status (Offset) 0x210 in PLB space)	0x0069	R	911
I2O0_ISTAT	I2O Status (Offset 0x00 in PLB space)	0x006A	R	912
	Reserved	0x006B		
I2O0_DMA_OP	DMA Observability Port (Offset 0x278 in PLB space)	0x006C	R	912
I2O0_SEAT	Slave Error Attribute Register (Offset 0x170 in PLB space)	0x006D	R	912
I2O0_SEAD	Slave Error Address Register (Offset 0x174 in PLB space)	0x006E	R	912
I2O0_I2O_OP	I2O Observability Port (Offset F0h in PLB space)	0x006F	R	913
I2O0_HWR0L	High Bandwidth PLB Write Interrupt Region 0 Base Low	0x0070	R/W	913
I2O0_HWR0H	High Bandwidth PLB Write Interrupt Region 0 Base High	0x0071	R/W	913
I2O0_HWROM	High Bandwidth PLB Write Interrupt Region 0 Mask/Attribute	0x0072	R/W	913
I2O0_HWR1L	High Bandwidth PLB Write Interrupt Region 1 Base Low	0x0073	R/W	914
I2O0_HWR1H	High Bandwidth PLB Write Interrupt Region 1 Base High	0x0074	R/W	914
I2O0_HWR1M	High Bandwidth PLB Write Interrupt Region 1 Mask/Attribute	0x0075	R/W	914
I2O0_MISCC	Miscellaneous Configuration	0x0076	R/W	915
	Reserved	0x0077 - 0x007F		n/a

26.6.1.1 Low Latency PLB Write Interrupt Region 0 Base Low Register (I2O0_LLIRQOL)

This register defines the lower 32 bits of a 64-bit address. The upper 32 bits are defined in the LL PLB Write Interrupt Region 0 Base High register. The address formed by these two registers is the starting address for the primary Write Interrupt Region on the Low Latency PLB. The following figure shows the bit definitions for I2O0_LLIRQ0L.

Figure 26-9. Low Latency PLB Write Interrupt Region 0 Base Low Register (I2O0_LLIRQ0L)

31:3	LLIRQ0L	Write Interrupt Region 0 Base Address Low	
2:0		Reserved.	

26.6.1.2 Low Latency PLB Write Interrupt Region 0 Base High Register (I2O0_LLIRQOH)

This register defines the upper 32 bits of a 64-bit address. The lower 32 bits are defined in the LL PLB Write Interrupt Region 0 Base Low register. The address formed by these two registers is the starting address for the primary Write Interrupt Region on the Low Latency PLB.

Figure 26-10. Low Latency PLB Write Interrupt Region 0 Base High Register (I2O0_LLIRQ0H)

31:0	LLIRQ0H	Write Interrupt Region 0 Base Address High	

26.6.1.3 Low Latency PLB Write Interrupt Region 0 Mask/Attribute Register (I2O0_LLWR0M)

This register defines the size of the primary Write Interrupt Region on the Low Latency PLB and specifies whether it is enabled. The size of the region is specified by setting bits in this register to 0. The contents of this register are logically ANDed with both the lower 32 bits of the programmed base address of the region and the lower 32 address bits presented on the PLB. If the upper 32 bits of the base address match the upper 32 bits presented on the PLB and the masked values are identical, then the address on PLB is said to be within the interrupt region. The following example demonstrates the relationship more thoroughly.

Assume the base address (lower 32 bits) is programmed to 0x0F3C_1000 and the mask is programmed to 0xFFFF_F000. If the address presented on PLB (lower 32 bits) is 0x0F3C_1FFF, then the transaction takes place within the region because:

0x0F3C_1000 (base address) and 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_1FFF (PLB address) and 0xFFFF_F000 (mask) = 0x0F3C_1000

The two masked values are identical; the transfer is within the region

If the PLB address presented were 0x0F3C_9000, then the transaction is outside the region:

0x0F3C_1000 (base address) and 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_9000 (PLB address) and 0xFFFF_F000 (mask) = 0x0F3C_9000

Note: The behavior of the mask requires the base address be aligned to the size of the boundary the mask defines (i.e. all 0s in the mask force the respective bits in the base address to 0).

There must not be any intermediate 1s between 0s in the mask (i.e. 0xFF0F_000 is not allowed).

Figure 26-11. Low Latency PLB Write Interrupt Region 0 Mask/Attribute Register (I2O0_LLWR0M)			
31-3	MASK	Mask	Determines which bits of the Write Interrupt Region 0 Base address register are compared with PLB addresses. The minimum region size is 8 bytes.
2-1		Reserved.	
0	ENBL	Enable	Enable interrupt for region 0

26.6.1.4 Low Latency PLB Write Interrupt Region 1 Base Low Register (I2O0_LLIRQ1L)

This register defines the lower 32 bits of a 64-bit address. The upper 32 bits are defined in the LL PLB Write Interrupt Region 1 Base High register. The address formed by these two registers is the starting address for the secondary Write Interrupt Region on the Low Latency PLB.

Figure 26-12. Low Latency PLB Write Interrupt Region 1 Base Low Register (I2O0_LLIRQ1L)			
31:3	LLIRQ1L	Write Interrupt Region 1 Base Address Low	
2:0		Reserved.	

26.6.1.5 Low Latency PLB Write Interrupt Region 1 Base High Register (I2O0_LLIRQ1H)

This register defines the upper 32 bits of a 64-bit address. The lower 32 bits are defined in the LL PLB Write Interrupt Region 1 Base Low register. The address formed by these two registers is the starting address for the secondary Write Interrupt Region on the Low Latency PLB.

Figure 26-13. Low Latency PLB Write Interrupt Region 0 Base High Register (I2O0_LLIRQ1H)			
31:0	LLIRQ1H	Write Interrupt Region 1 Base Address High	

26.6.1.6 Low Latency PLB Write Interrupt Region 1 Mask/Attribute Register (I2O0_LLWR1M)

This register defines the size of the secondary Write Interrupt Region on the Low Latency PLB. The size of the region is specified by setting bits in this register to 0. The contents of this register are logically ANDed with both the lower 32 bits of the programmed base address of the region and the lower 32 address bits presented on the PLB. If the upper 32 bits of the base address match the upper 32 bits presented on the PLB and the masked values are identical, then the address on PLB is said to be within the interrupt region. The following example demonstrates the relationship more thoroughly.

Assume the base address (lower 32 bits) is programmed to 0x0F3C_1000 and the mask is programmed to 0xFFFF_F000. If the address presented on PLB (lower 32 bits) is 0x0F3C_1FFF, then the transaction takes place within the region because:

0x0F3C_1000 (base address) & 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_1FFF (PLB address) & 0xFFFF_F000 (mask) = 0x0F3C_1000

The two masked values are identical; the transfer is within the region

If the PLB address presented were 0x0F3C_9000, then the transaction is outside the region:

0x0F3C_1000 (base address) & 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_9000 (PLB address) & 0xFFFF_F000 (mask) = 0x0F3C_9000

Note: The behavior of the mask requires the base address be aligned to the size of the boundary the mask defines (i.e. all 0s in the mask force the respective bits in the base address to 0).

There must be no intermediate 1s between 0s in the mask (i.e. 0xFF0F_000 is not allowed).

.

Figure 26-14. Low Latency PLB Write Interrupt Region 1 Mask/Attribute Register (I2O0_LLWR1M)

31-3	MASK	Mask	Determines which bits of the Write Interrupt Region 1 Base address register are compared with PLB addresses. The minimum region size is eight bytes.
2-1		Reserved.	
0	ENBL	Enable	Enable interrupt for region 1

26.6.1.7 I2O Base Address Low Register (I2O0_IBAL)

This register defines the lower 32 bits of the base address for I2O registers. Only the most significant 20 bits are writable, causing the base address to start on a 4KB boundary. The upper 32 bits are defined in I2O Base Address High. It is recommended that I2O Base Address High be programmed prior to setting the enable bit in this register.

Figure 26-15. I2O Base Address Low Register (I2O0_IBAL)

31-12	RIBL	I2O/DMA Register Interface Base Address Low	
11-1		Reserved.	
0	ENBL	I2O/DMA Register Enable	When set to 1 this bit enables I2O/DMA register access. This bit should only be set to one when both the low and the high address registers are programmed.

26.6.1.8 I2O Base Address High Register (I2O0_IBAH)

This register defines the upper 32 bits of the base address for I2O registers. This register should be programmed before setting the enable bit in the I2O Base Address Low register.

Figure 26-16. I2O Base Address High Register (I2O0_IBAH)

31:0	IBAH	I2O/DMA Register Interface Base Address High	

26.6.1.9 DMA Status Register (I2O0_DMA_ST)

This register can be used to read the PLB-mapped DMA Port Status register by means of the DCR interface.

Figure 26-17. DMA Status Register (I2O0_DMA_ST)			
31-21		Reserved.	
20-0	DSTS	DMA Status	This is a copy of bits 20:0 of the DMA Status register (I2O0_DMA_DSTS)

26.6.1.10 I2O Status Register (I2O0_ISTAT)

This register can be used to read the PLB-mapped I2O Status register by means of the DCR interface.

Figure 26-18. I2O Status Register (I2O0_ISTAT)			
31:10		Reserved.	
9:0	ISTS	I2O Status	This is a copy of bits 9:0 of the I2O Status register (I2O0_ISTS)

26.6.1.11 DMA Observability Port (I2O0_DMA_OP)

This register can be used to read the PLB-mapped DMA Observability Port for DMA Port by means of the DCR interface.

Figure 26-19. DMA Observability Port (I2O0_DMA_OP)			
31:0	DOPP	DMA Observability Port	This is a copy of bits 31:0 of the DMA Observability Port (I2O0_DMA_OP)

26.6.1.12 Slave Error Attribute Register (I2O0_SEAT)

This register can be used to read the PLB-mapped Slave Error Attribute Register by means of the DCR interface.

Figure 26-20. DMA Slave Error Attribute Register (I2O0_SEAT)			
31:0	SEATR	Slave Error Attribute Register	This is a copy of bits 31:0 of the Slave Error Attribute Register (I2O0_DMAx_SEAT)

26.6.1.13 Slave Error Address Register (I2O0_SEAD)

This register can be used to read the PLB-mapped Slave Error Attribute Register by means of the DCR interface.

Figure 26-21. DMA Slave Error Address Register (I2O0_SEAD)			
31:0	SEADR	Slave Error Address Register	This is a copy of bits 31:0 of the Slave Error Address Register (I2O0_DMAx_SEAD)

26.6.1.14 I2O Observability Port (I2O0_I2O_OP)

This register can be used to read the PLB-mapped I2O Observability Port Register through the DCR interface. See *I2O Observability Port Register (I2O0_IOPT)* on page 946 for a description of the contents of the PLB-mapped I2O Observability Port Register.

Figure 26-22. I2O Observability Port (I2O0_I2O_OP)

31:0	IOPRT	I2O Observability Port	This is a copy of bits 31:0 of the I2O Observability Port Register.

26.6.1.15 High Bandwidth PLB Write Interrupt Region 0 Base Low Register (I2O0_HWR0L)

This register defines the lower 32 bits of a 64-bit address. The upper 32 bits are defined in the HB PLB Write Interrupt Region 0 Base High register. The address formed by these two registers is the starting address for the primary Write Interrupt Region on the High Bandwidth PLB.

Figure 26-23. High Bandwidth PLB Write Interrupt Region 0 Base Low Register (I2O0_HWR0L)

31:3	HWR0L	Write Interrupt Region 0 Base Address Low	
2:0		Reserved.	

26.6.1.16 High Bandwidth PLB Write Interrupt Region 0 Base High Register (I2O0_HWR0H)

This register defines the upper 32 bits of a 64-bit address. The lower 32 bits are defined in the HB PLB Write Interrupt Region 0 Base Low register. The address formed by these two registers is the starting address for the primary Write Interrupt Region on the High Bandwidth PLB.

Figure 26-24. High Bandwidth PLB Write Interrupt Region 0 Base High Register (I2O0_HWR0H)

31:0	HWR0H	Write Interrupt Region 0 Base address High	

26.6.1.17 High Bandwidth PLB Write Interrupt Region 0 Mask/Attribute Register (I2O0_HWR0M)

This register defines the size of the primary Write Interrupt Region on the High Bandwidth PLB. The size of the region is specified by setting bits in this register to 0. The contents of this register are logically ANDed with both the lower 32 bits of the programmed base address of the region and the lower 32 address bits presented on the PLB. If the upper 32 bits of the base address match the upper 32 bits presented on the PLB and the masked values are identical, then the address on PLB is said to be within the interrupt region. The following example demonstrates the relationship more thoroughly.

Assume the base address (lower 32 bits) is programmed to 0x0F3C_1000 and the mask is programmed to 0xFFFF_F000. If the address presented on PLB (lower 32 bits) is 0x0F3C_1FFF, then the transaction takes place within the region because:

0x0F3C_1000 (base address) & 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_1FFF (PLB address) & 0xFFFF_F000 (mask) = 0x0F3C_1000

The two masked values are identical; the transfer is within the region

If the PLB address presented were 0x0F3C_9000, then the transaction is outside the region:

0x0F3C_1000 (base address) & 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_9000 (PLB address) & 0xFFFF_F000 (mask) = 0x0F3C_9000

Notes:

1. The behavior of the mask requires that the base address be aligned to the size of the boundary the mask defines (i.e. all 0's in the mask force the respective bits in the base address to 0).
2. There must be no intermediate 1's between 0's in the mask (i.e. 0xFF0F_000 is not allowed).

Figure 26-25. High Bandwidth PLB Write Interrupt Region 0 Mask/Attribute Register (I2O0_HWR0M)

31:3	MASK	Mask	Determines which bits of the Write Interrupt Region Base address register are compared with PLB addresses. The minimum region size is 8 bytes.
2:1		Reserved.	
0	ENBL	Enable	Enable interrupt for region 0

26.6.1.18 High Bandwidth PLB Write Interrupt Region 1 Base Low Register (I2O0_HWR1L)

This register defines the lower 32 bits of a 64-bit address. The upper 32 bits are defined in the HB PLB Write Interrupt Region 1 Base High register. The address formed by these two registers is the starting address for the secondary Write Interrupt Region on the High Bandwidth PLB.

Figure 26-26. High Bandwidth PLB Write Interrupt Region 1 Base Low Register (I2O0_HWR1L)

31:3	HWR1L	Write Interrupt Region Base Address Low	
2:0		Reserved.	

26.6.1.19 High Bandwidth PLB Write Interrupt Region 1 Base High Register (I2O0_HWR1H)

This register defines the upper 32 bits of a 64-bit address. The lower 32 bits are defined in the HB PLB Write Interrupt Region 1 Base Low register. The address formed by these two registers is the starting address for the secondary Write Interrupt Region on the High Bandwidth PLB.

Figure 26-27. High Bandwidth PLB Write Interrupt Region 1 Base High Register (I2O0_HWR1H)

31:0	HWR1H	Write Interrupt Region 1 Base Address High	

26.6.1.20 High Bandwidth PLB Write Interrupt Region 1 Mask/Attribute Register (I2O0_HWR1M)

This register defines the size of the secondary Write Interrupt Region on the High Bandwidth PLB. The size of the region is specified by setting bits in this register to 0. The contents of this register are logically ANDed with both the lower 32 bits of the programmed base address of the region as well as the lower 32 address bits presented on the PLB. If the upper 32 bits of the base address match the upper 32 bits presented on the PLB and the masked, lower 32-bit values are identical, then the address on PLB is considered to be within the interrupt region. The following example demonstrates the relationship more thoroughly.

Assume the base address (lower 32 bits) is programmed to 0x0F3C_1000 and the mask is programmed to 0xFFFF_F000. If the address presented on PLB (lower 32 bits) is 0x0F3C_1FFF, then the transaction takes place within the region because:

0x0F3C_1000 (base address) & 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_1FFF (PLB address) & 0xFFFF_F000 (mask) = 0x0F3C_1000

The two masked values are identical; the transfer is within the region

If the PLB address presented were 0x0F3C_9000, then the transaction is outside the region:

0x0F3C_1000 (base address) & 0xFFFF_F000 (mask) = 0x0F3C_1000
0x0F3C_9000 (PLB address) & 0xFFFF_F000 (mask) = 0x0F3C_9000

Note: The behavior of the mask requires the base address be aligned to the size of the boundary the mask defines (i.e. all 0's in the mask force the respective bits in the base address to 0).

There must be no intermediate 1's between 0's in the mask (i.e. 0xFF0F_000 is not allowed)

.

Figure 26-28. High Bandwidth PLB Write Interrupt Region 1 Mask/Attribute Register (I2O0_HWR1M)

31:3	MASK	Mask	Determines which bits of the Write Interrupt Region 1 Base address register are compared with PLB addresses. The minimum region size is 8 bytes.
2:1		Reserved.	
0	ENBL	Enable	Enable interrupt for region 1

26.6.1.21 Miscellaneous Configuration Register (I2O0_MISCC)

This register contains configuration options that do not belong in any of the previous register definitions.

Figure 26-29. Miscellaneous Configuration Register (I2O0_MISCC)

31:1		Reserved.	
0	PXLAT	MFA Address Modification: When set to 1, the upper bit (63) of the pull MFA is forced to 1.	This special function is used to redirect the MFA to some specific address. It is not part of the I2O specification.

26.6.2 Memory Mapped DMA Registers

This section describes the DMA registers that are memory mapped registers. The following register summary includes the specific memory addresses for the individual registers prefixed and I2O0_DMA_. The detail register descriptions that follow the summary contain one description prefixed I2O0_DMAx_ for each register from the I2O0_DMA_ group of registers.

These registers are associated with on-chip peripherals and are mapped into the system memory (PLB) space. They are accessed through load/store instructions that contain the register addresses.

The PLB Slave is programmed to claim a 4-KB I2O window for these registers. All address offsets are from the start of that window (address 0x4_0010_0000 defined in the Base Address registers I2O0_IBAL and I2O0_IBAH.

All reserved registers are read-only and return zeroes. The following table summarizes the available registers.

Table 26-25. Memory Mapped DMA Register Summary

Mnemonic	Register	Address	Access	Page
I2O0_DMA_CPFPL	DMA Command Pointer FIFO Port Low	0x4 0010 0200	W	917
I2O0_DMA_CPFPH	DMA Command Pointer FIFO Port High	0x4 0010 0204	W	917
I2O0_DMA_CSFPL	DMA Completion Status FIFO Port Low	0x4 0010 0208	R	917
I2O0_DMA_CSFPH	DMA Completion Status FIFO Port High	0x4 0010 020C	R	918
I2O0_DMA_DSTS	DMA Status (copy at 0x0 0069 in DCR space)	0x4 0010 0210	R/W/Clear (specific bits)	918
I2O0_DMA_CFG	DMA Configuration	0x4 0010 0214	R/W	920
	Reserved.	0x4 0010 0218 - 021F		n/a
I2O0_DMA_CPFHP	DMA Command Pointer FIFO Head Pointer	0x4 0010 0220	R/W	921
I2O0_DMA_CPFTP	DMA Command Pointer FIFO Tail Pointer	0x4 0010 0222	R/W	921
I2O0_DMA_CSFHP	DMA Completion Status FIFO Head Pointer	0x4 0010 0224	R/W	921
I2O0_DMA_CSFTP	DMA Completion Status FIFO Tail Pointer	0x4 0010 0226	R/W	922
	Reserved.	0x4 0010 0228 - 022F		n/a
I2O0_DMA_ACPL	DMA Active Command Pointer Low	0x4 0010 0230	R	922
I2O0_DMA_ACPH	DMA Active Command Pointer High	0x4 0010 0234	R	923
I2O0_DMA_S1BPL	DMA SG1 Buffer Pointer Low	0x4 0010 0238	R	923
I2O0_DMA_S1BPH	DMA SG1 Buffer Pointer High	0x4 0010 023C	R	923
I2O0_DMA_S2BPL	DMA SG2 Buffer Pointer Low	0x4 0010 0240	R	923
I2O0_DMA_S2BPH	DMA SG2 Buffer Pointer High	0x4 0010 0244	R	923
I2O0_DMA_S3BPL	DMA SG3 Buffer Pointer Low	0x4 0010 0248	R	924
I2O0_DMA_S3BPH	DMA SG3 Buffer Pointer High	0x4 0010 024C	R	924
	Reserved.	0x4 0010 0250 - 025F		n/a
I2O0_DMA_EARL	DMA Error Address Register Low	0x4 0010 0260	R/W/Clear	924
I2O0_DMA_EARH	DMA Error Address Register High	0x4 0010 0264	R/W/Clear	924
	Reserved.	0x4 0010 0268 - 026F		n/a
I2O0_DMA_SEAT	DMA Slave Error Attribute Register (copy at 0x0 006D in DCR space)	0x4 0010 0270	R	925
I2O0_DMA_SEAD	DMA Slave Error Address Register (copy at 0x0 006E in DCR space)	0x4 0010 0274	R	925
I2O0_DMA_OP	DMA Observability Port	0x4 0010 0278	R	926
I2O0_DMA_FSIZ	DMA FIFO Size	0x4 0010 027C	R/W	927
	Reserved.	0x4 0010 0280 - 02FF		n/a

26.6.2.1 DMA Command Pointer FIFO Port Low Register (I2O0_DMAx_CPFPL)

This register is used to access the DMA Command Descriptor Block (CDB) Pointer FIFO. The value written to this register is placed at the head of the CDB Pointer FIFO and represents the lower half (32 bits) of a CDB pointer.

All pointers are stored internally as 64-bit pointers regardless of the size indicated. For 32-bit pointers, this means the upper 32 bits are automatically filled with zeroes. Writes to this register less than 32 bits in width or when the FIFO is full will generate an error.

Figure 26-30. DMA Command Pointer FIFO Port Low Register (I2O0_DMAx_CPFPL)

31:4	CDBP	CDB Pointer Address Low	Most significant 28 bits of a 32-bit CDB pointer; implies a CDB must reside on a 16-byte boundary
3	NOINT	No Interrupt: 0 Generate Interrupt when complete 1 Do not generate an interrupt when complete	
2	ADSZ	Address Size: 0 CDB pointer is 32 bits wide 1 CDB pointer is 64 bits wide	
1:0		Reserved.	

26.6.2.2 DMA Command Pointer FIFO Port High Register (I2O0_DMAx_CPFPH)

This register is used to access the DMA Command Descriptor Block (CDB) Pointer FIFO. The value written to this register is placed at the head of the CDB Pointer FIFO and represents the upper half (32 bits) of a CDB pointer.

Writes to this register before writing the Command Pointer FIFO Port (Lower 32 Bits) with a value indicating a 64-bit Command Pointer will result in an error.

Writes to this register less than 32 bits in width or when the FIFO is full will result in an error.

Figure 26-31. DMA Command Pointer FIFO Port High Register (I2O0_DMAx_CPFPH)

31:0	CPFPH	CDB Pointer Address High	Most significant 32 bits of a 64-bit CDB pointer.

26.6.2.3 DMA Completion Status FIFO Port Low Register (I2O0_DMAx_CSFPL)

This register is used to access the DMA Command Descriptor Block Pointer Status FIFO. The value read of this register represents the lower 32 bits of a Completion Status entry. Command execution status, the size of the command's address (32 or 64-bit), and whether this entry generated an interrupt are all reflected in the lower/least significant bits of this register.

Reads of less than 32 bits will result in an error. Reads will return zeroes if the FIFO is empty; any other value represents a valid Completion Status entry.

Figure 26-32. DMA Completion Status FIFO Port Low Register (I2O0_DMAx_CSFPL)			
31:4	CDBPL	Address of a Completed CDB	Most significant 28 bits of a completed 32-bit CDB pointer
3	NOINT	No Interrupt: 0 This CDB did not generate an interrupt 1 This CDB generated an interrupt	
2	ADSIZ	Address Size: 0 CDB pointer is 32 bits wide 1 CDB pointer is 64 bits wide	
1:0	CSTAT	Status: 00 DMA operation completed normally 01 Data Check returned with a miscompare 10 Error fetching from Command Status FIFO (address size and field are invalid) 11 CDB encountered an error, but did not halt the engine	Completion status of this command

26.6.2.4 DMA Completion Status FIFO Port High Register (I2O0_DMAx_CSFPH)

This register is used to access the DMA Command Descriptor Block Pointer Status FIFO. This value represents the upper half (32 bits) of a Completion Status entry.

This register can be read only if a previous read of the Completion Status FIFO Port (Lower 32 Bits) indicates the Completion Status entry is 64 bits wide. Attempting to read this register when a 32-bit entry is indicated will result in an error.

Reads less than 32 bits wide will result in an error.

Figure 26-33. DMA Completion Status FIFO Port High Register (I2O0_DMAx_CSFPH)			
31:0	CSTAT	Address of a Completed CDB	Most significant 32 bits of a 64-bit pointer to a completed CDB

26.6.2.5 DMA Status Register (I2O0_DMAx_DSTS)

This register is used to report the status of the DMA logic. It is cleared to 0 during a system reset. All error conditions are logically ORed together and forwarded as a single DMA Error Interrupt to the 460EX/EXr/GT.

Figure 26-34. DMA Status Register (I2O0_DMAx_DSTS)			
31:21		Reserved.	
20:19	FAERR	FIFO Access Error: 00 Command Pointer FIFO read 01 Command Pointer FIFO write 10 Completion Status FIFO read 11 Completion Status FIFO write Note: The values in this register are valid only when the DMA controller PLB Error contains non-normal status.	

18:16	FMERR	Status of the DMA FIFO Manager PLB Master: 000 Normal status 001 Wrong PLB Slave size 010 PLB MrdErr or PLB MWrErr 011 PLB MIRQ asserted while transfer in progress 100 PLB time-out asserted while waiting for address acknowledge Note: In cases where a DMA Controller PLB Error halts the DMA engine, writing a 1 to the DMA Halted bit also clears these bits.	
15:13	RSERR	Status of the PLB Slave: 000 Normal status 001 Rd/Wr to a reserved address or invalid byte enabled 010 Write to a full FIFO (can be set by I2O access) 011 A read or write to a DMA FIFO that does not exist (can be set by I2O access) 101 High Low Port Order Error (Set when upper and lower 32 bits of the CP or CS ports are accessed out of order) Note: Writing a 1 to any of these bits clears all three. In addition, it clears the Slave Error Attribute and Address registers.	
12:10	PWERR	PLB Write Error: Status of the DMA Data Transfer Engine PLB Master: 000 Normal status 001 Wrong PLB slave size 010 PLB MWrErr 011 PLB MIRQ asserted while transfer in progress 100 PLB time-out asserted while waiting for address acknowledge. Note: Writing a 1 to the DMA Halted bit clears all three bits.	
9:7	PRERR	PLB Read Error: Status of the DMA Data Transfer Engine PLB Master: 000 Normal status 001 Wrong PLB slave size 010 PLB MRdErr 011 PLB MIRQ asserted while transfer in progress 100 PLB time-out asserted while waiting for address acknowledge. Note: Writing a 1 to the DMA Halted bit clears all three bits.	
6	DHALT	DMA Halted This bit is cleared by writing a 1 to it or on a System Reset	Asserted when an error causes the DMA Data Transfer Engine to halt during a DMA operation. The following errors can halt the DMA Data Transfer Engine: DMA Controller PLB Master Error on CDB pointer reads CDB Status writes DMA Engine PLB Master Error on reads or writes. Illegal opcode
5	PMIRQ	PLB MIRQ Asserted This bit is cleared when a 1 is written to the DMA Halted bit.	
4	VALER	Valid Error	Indicates the error address high/low registers contain valid addresses.

3	ILLOP	Illegal opcode decoded from a CDB	
2:0		Reserved.	

26.6.2.6 DMA Configuration Register (I2O0_DMAx_CFG)

The following figure defines the bits that configure the DMA data transfer engine.

Figure 26-35. DMA Configuration Register (I2O0_DMAx_CFG)

31	DXEPD	DMA Data Transfer Engine PLB Prefetch Disable: 0 DMA Data Transfer Engine PLB master can request that the PCI interface prefetch data when appropriate 1 DMA Data Transfer Engine PLB Master cannot request that the PCI interface prefetch data on its behalf.	
30	DXERO	DMA Data Transfer Engine PLB relaxed ordering: 0 DMA Data Transfer Engine PLB Master always requests ordered write transfers. 1 DMA Data Transfer Engine PLB Master can request ordered write transfers.	
29:28		Reserved.	
27:26	DXEPR	DMA Data Transfer Engine PLB Priority: 11 Highest Priority 10 Next Highest Priority 01 Next Highest Priority 00 Lowest Priority	The value of this register is asserted on the PLB priority signals when conducting a transaction on behalf of the DMA Engine.
25		Reserved.	
24:23	DFMPP	DMA FIFO Manager PLB Priority: 11 Highest Priority 10 Next Highest Priority 01 Next Highest Priority 00 Lowest Priority	The value of this register is asserted on the PLB priority signals when conducting a transaction on behalf of the DMA Control logic.
22:21		Reserved.	
20	NOHLT	No Halt on Error: 0 DMA Data Transfer Engine halts when haltable errors occur 1 DMA Data Transfer Engine does not halt when an error occurs	
19	FALGN	Force 64-byte Alignment: 0 The DMA Data Transfer Engine attempts to align itself to a 16-byte address before it starts to break the transfer into 512-byte bursts. 1 The DMA Data Transfer Engine attempts to align itself to a 64-byte address before it starts to break the transfer into 512-byte bursts.	
18:0		Reserved.	

26.6.2.7 DMA Command Pointer FIFO Head Pointer (I2O0_DMAx_CPFHP)

This is the actual head pointer used in maintaining the Command Pointer FIFO. Since this register is updated automatically during normal operation, caution should be used when writing this register. The user must ensure there are no accesses to the FIFO before directly writing to the FIFO pointer. Unpredictable behavior can result if a write occurs at the same time an automatic update takes place. As a safety measure, a write will be ignored if a request is pending on the DMA Back End. In any event, a read should be conducted to verify any attempted writes.

Figure 26-36. DMA Command Pointer FIFO Head Pointer (I2O0_DMAx_CPFHP)

15:14		Reserved.	
13:0	CPFHP	Command Pointer FIFO Head Pointer	The position indicated is a byte offset from the base address for the FIFO. The FIFO Head Pointer has 8-byte granularity, thus the lower 3 bits, 2-0 are always read as zero.

26.6.2.8 DMA Command Pointer FIFO Tail Pointer (I2O0_DMAx_CPFTP)

This is the actual tail pointer used in maintaining the Command Pointer FIFO. Since this register is updated automatically during normal operation, caution should be used when writing this register. The user must ensure there are no accesses to the FIFO before directly writing to the FIFO pointer. Unpredictable behavior can result if a write occurs at the same time an automatic update takes place. As a safety measure, a write will be ignored if a request is pending on the DMA Back End. In any event, a read should be conducted to verify any attempted writes.

Figure 26-37. DMA Command Pointer FIFO Tail Pointer (I2O0_DMAx_CPFTP)

15		Reserved.	
14	FIFOF	FIFO Full: 0 FIFO is not full 1 FIFO is full	
13:0	CPFTP	Command Pointer FIFO Tail Pointer	The position indicated is a byte offset from the base address for the FIFO. The FIFO Head Pointer has 8-byte granularity, thus the lower 3 bits, 2-0 are always read as zero.

26.6.2.9 DMA Completion Status FIFO Head Pointer (I2O0_DMAx_CSFHP)

This is the actual head pointer used in maintaining the Completion Status FIFO. Since this register is updated automatically during normal operation, caution should be used when writing this register. The user must ensure there are no accesses to the FIFO before directly writing to the FIFO pointer. Unpredictable behavior can result if a write occurs at the same time an automatic update takes place. As a safety measure, a write will be ignored if a request is pending on the DMA Back End. In any event, a read should be conducted to verify any attempted writes.

Note: A write of the Completion Status FIFO Head Pointer will disrupt the internal logic for interrupt generation. Therefore, the logic will reset itself to the case of an empty FIFO. Consequently, the pointer write must guarantee the FIFO is left in the empty to avoid unpredictable interrupt generation.

Figure 26-38. DMA Completion Status FIFO Head Pointer (I2O0_DMAx_CSFHP)			
15:14		Reserved.	
13:0	CSFHP	Command Status FIFO Head Pointer	These read/write bits indicate the current position of the head pointer within the FIFO. The position indicated is a byte offset from the FIFO base address. The FIFO Head Pointer has 8-byte granularity, thus the lower 3 bits, 2-0 are always read as zero.

26.6.2.10 DMA Completion Status FIFO Tail Pointer (I2O0_DMAx_CSFTP)

This is the actual tail pointer used in maintaining the Completion Status FIFO. Since this register is updated automatically during normal operation, caution should be used when writing this register. The user must ensure there are no accesses to the FIFO before directly writing to the FIFO pointer. Unpredictable behavior can result if a write occurs at the same time an automatic update takes place. As a safety measure, a write will be ignored if a request is pending on the DMA Back End. In any event, a read should be conducted to verify any attempted writes.

Note: A write of the Completion Status FIFO Head Pointer will disrupt the internal logic for interrupt generation. Therefore, the logic will reset itself to the case of an empty FIFO. Consequently, the pointer write must guarantee the FIFO is left in the empty to avoid unpredictable interrupt generation.

Figure 26-39. DMA Completion Status FIFO Tail Pointer (I2O0_DMAx_CSFTP)			
15		Reserved.	
14	FIFOF	FIFO Full: 0 FIFO is not full 1 FIFO is full	
13:0	CSFTP	Completion Status FIFO Tail Pointer	The position indicated is a byte offset from the base address for the FIFO. The FIFO tail Pointer has 8-byte granularity, thus the lower 3 bits, 2-0 are always read as zero.

26.6.2.11 DMA Active Command Pointer Low (I2O0_DMAx_ACPL)

Figure 26-40. DMA Active Command Pointer Low (I2O0_DMAx_ACPL)			
31:4	ACPL	Active Command Pointer	Contains the most significant 28 bits of a 32-bit pointer to the CDB currently being processed. The bits are cleared at reset.
3	NOINT	Address Size: 0 CDB generates an interrupt on completion 1 CDB does not generate an interrupt on completion	
2	ADSIZ	Address Size: 0 This pointer is 32 bits wide 1 This pointer is 64 bits wide	
1:0		Reserved.	

26.6.2.12 DMA Active Command Pointer High (I2O0_DMAx_ACPH)

Figure 26-41. DMA Active Command Pointer High (I2O0_DMAx_ACPH)			
31:0	ACPH	Active Command Pointer High	Contains the most significant 32 bits of a 64-bit pointer to the CDB currently being processed. These bits are cleared on reset.

26.6.2.13 DMA SG1 Buffer Pointer Low (I2O0_DMAx_S1BPL)

.

Figure 26-42. DMA SG1 Buffer Pointer Low (I2O0_DMAx_S1BPL)			
31:0	S1BPL	SG1 Buffer Pointer Low	Contains the least significant 32 bits of the 64-bit SG1 pointer for the last CDB read. This value may not be the same as the active CDB's SG1 buffer pointer due to CDB prefetching. This register is cleared at reset.

26.6.2.14 DMA SG1 Buffer Pointer High (I2O0_DMAx_S1BPH)

.

Figure 26-43. DMA SG1 Buffer Pointer High (I2O0_DMAx_S1BPH)			
31:0	S1BPH	SG1 Buffer Pointer High	Contains the most significant 32 bits of the 64-bit SG1 pointer for the last CDB read. This value may not be the same as the active CDB's SG1 buffer pointer due to CDB prefetching. This register is cleared at reset.

26.6.2.15 DMA SG2 Buffer Pointer Low (I2O0_DMAx_S2BPL)

.

Figure 26-44. DMA SG2 Buffer Pointer Low (I2O0_DMAx_S2BPL)			
31:0	S2BPL	SG2 Buffer Pointer Low	Contains the least significant 32 bits of the 64-bit SG2 pointer for the last CDB read. This value may not be the same as the active CDB's SG2 buffer pointer due to CDB prefetching. This register is cleared at reset.

26.6.2.16 DMA SG2 Buffer Pointer High (I2O0_DMAx_S2BPH)

.

Figure 26-45. DMA SG2 Buffer Pointer High (I2O0_DMAx_S2BPH)			
31:0	S2BPH	SG2 Buffer Pointer High	Contains the most significant 32 bits of the 64-bit SG2 pointer for the last CDB read. This value may not be the same as the active CDB's SG2 buffer pointer due to CDB prefetching. This register is cleared at reset.

26.6.2.17 DMA SG3 Buffer Pointer Low (I2O0_DMAx_S3BPL)

.

Figure 26-46. DMA SG3 Buffer Pointer Low (I2O0_DMAx_S3BPL)			
31:0	S3BPL	SG3 Buffer Pointer Low	Contains the least significant 32 bits of the 64-bit SG3 pointer for the last CDB read. This value may not be the same as the active CDB's SG3 buffer pointer due to CDB prefetching. This register is cleared at reset.

26.6.2.18 DMA SG3 Buffer Pointer High (I2O0_DMAx_S3BPH)

.

Figure 26-47. DMA SG3 Buffer Pointer High (I2O0_DMAx_S3BPH)			
31:0	S3BPH	SG3 Buffer Pointer High	Contains the most significant 32 bits of the 64-bit SG3 pointer for the last CDB read. This value may not be the same as the active CDB's SG3 buffer pointer due to CDB prefetching. This register is cleared at reset.

26.6.2.19 DMA Error Address Low Register (I2O0_DMAx_EARL)

This register contains the lower 32 bits of a 64-bit address that encountered an error during command execution. The address recorded may not be the exact address at which the failure occurred; it is simply the beginning address of the transfer that encountered the error (in the case of a burst). If the No Halt On Error bit (in DMA Configuration register) is set, then this register records only the first address that encountered an error. Any further error addresses are lost.

This register and the Error Address Register (Upper 32 Bits) are cleared in any of three situations:
- Writing a 1 to any bit of this register
- Writing a 1 to any bit of the Error Address Register (Upper 32 Bits)
- Writing a 1 to the DMA Halted bit (in DMA Status register)

Figure 26-48. DMA Error Address Low Register (I2O0_DMAx_EARL)			
31:0	EARL	Error Address Register	This register contains the lower 32 bits of an address that generated an error during command execution.

26.6.2.20 DMA Error Address High Register (I2O0_DMAx_EARH)

This register contains the upper 32 bits of a 64-bit address that encountered an error during command execution. The address recorded may not be the exact address at which the failure occurred; it is simply the beginning address of the transfer that encountered the error (in the case of a burst). If the No Halt On Error bit (in DMA Configuration register) is set, then this register records only the first address that encountered an error. Any further error addresses are lost.

This register and the Error Address Register (Lower 32 Bits) are cleared in any of three situations:
- Writing a 1 to any bit of this register
- Writing a 1 to any bit of the Error Address Register (Lower 32 Bits)
- Writing a 1 to the DMA Halted bit (in DMA Status register)

Figure 26-49. DMA Error Address High Register (I2O0_DMAx_EARH)			
31:0	EARH	Error Address Register High	This register contains the upper 32 bits of an address that generated an error during command execution.

26.6.2.21 DMA Slave Error Attribute Register (I2O0_DMAx_SEAT)

The Master ID, Master Size, and BE bits of the Slave Error Attributes Register reflect attributes of the latest transfer the PLB Slave has address acknowledged on the PLB bus while the Error Transfer Type has normal status. When the PLB Slave detects an error, the register locks the attributes of the first error detected and remains locked. The Slave Error Attribute Register can be unlocked by clearing PLB Slave status bits of the I2O or DMA Status Registers.

Note: This address accesses a single Slave Error Attribute Register, visible from both I2O and DMA Register Interfaces.

Figure 26-50. DMA Slave Error Attribute Register (I2O0_DMAx_SEAT)			
31:25		Reserved.	
24:22	ERXTY	Error Transfer Type: 000 Normal status 010 Read transfer error 011 Read transfer aborted 100 Write transfer error 101 Write transfer aborted	The type of transfer that caused the PLB slave error.
21:18	MID	Master ID	Master ID of the PLB transfer.
17:16	MSIZ	Master Size	Size of the PLB master.
15:0	BEX	Byte Enables for Transfer	

26.6.2.22 DMA Slave Error Address Register (I2O0_DMAx_SEAD)

The Slave Error Address reflects the address of the latest transfer the PLB Slave has address acknowledged on the PLB bus while the Error Transfer Type bits of the Slave Error Attribute Register has normal status. When the PLB Slave detects an error, the register locks the address of the first error detected and remains locked. Clearing the PLB Slave status bits of the I2O or DMA Status Registers will unlock the Slave Error Address Register.

Note: This address accesses a single Slave Error Address Register, visible from both I2O and DMA Register Interfaces.

.

Figure 26-51. DMA Slave Error Address Register (I2O0_DMAx_SEAD)			
31:0	SEADR	Slave Error Address	The PLB slave error address contains the address associated with an error in accessing the I2O/DMA registers.

26.6.2.23 DMA Observability Port (I2O0_DMAx_OP)

Figure 26-52. DMA Observability Port (I2O0_DMAx_OP)

31:29	ECSMS	DMA Engine Control State Machine State	
28	ECSWP	DMA Engine - Completion Status Write Pending	
27:26	MCMSD	PLB Bus Master Control State Machine State (Drain)	
25:23	MCCXS	PLB Bus Master Control Current Transfer State (Drain)	
22:21	MCSMS	PLB Bus Master Control State Machine State (Fill)	
20:18	MCXTF	PLB Bus Master Control Current Transfer Type (Fill)	
17:15	EPMAS	DMA Engine PLB Master Address State Machine State	
14:13	EPMRS	DMA Engine PLB Master Read State Machine State	
12:11	EPMWS	DMA Engine PLB Master Write State Machine State	
10:9	EPMCX	DMA Engine PLB Master Control Current Transfer Type (Write)	
8:7	EPMFF	DMA Engine PLB Master Front State Machine state (Fill)	
6:5	EPMFD	DMA Engine PLB Master Front State Machine State (Drain)	
4	EPMAR	DMA Engine PLB Master Arbitration State Machine State	
3	PFUPS	DMA Data Path FIFO Underrun (Primary/Secondary)	
2	PFOPS	DMA Data Path FIFO Overflow (Primary/Secondary)	
1:0	RBETQ	DMA Register - (Back End) TQ State Machine State	

26.6.2.24 DMA FIFO Size Register (I2O0_DMAx_FSIZ)

Figure 26-53. DMA FIFO Size Register (I2O0_DMAx_FSIZ)			
31:13		Reserved.	
12	DFENB	DMA FIFO Enable: 0 DMA CDB and status FIFO do not exist 1 DMA CDB and status FIFO exist	If bit is set to 1, I2O memory space is reserved for the DMA Command Descriptor Block Pointer and Command Status FIFOs.
11		Reserved.	
10:0	DFSZ	DMA FIFO Size	This register can be loaded with any value 0 to 2047. The value plus 1 indicates the number of CDB pointers in the DMA Command Pointer and Completion Status FIFOs.

26.6.3 Memory Mapped I2O Registers

This section describes the I2O registers that are memory mapped registers. These registers are associated with on-chip peripherals and are mapped into the system memory (PLB) space. They are accessed through load/store instructions that contain the register addresses.

The PLB Slave is programmed to claim a 4KB I2O window for these registers. All address offsets are from the start of that window. All reserved registers are read-only and return zeros.

Reads and writes to this memory space must be single 64-bit (or less) accesses.

The table below summarizes the I2O messaging. Detailed descriptions follow the summary. The base address for these registers resides in PLB space and is defined in the I2O Base Address register. See *I2O Base Address Low Register (I2O0_IBAL)* on page 911 and *I2O Base Address High Register (I2O0_IBAH)* on page 911.

Table 26-26. I2O Memory Mapped Register Summary

Mnemonic	Register	Address	Access	Page
I2O0_ISTS	I2O Status (Also at address 0x006A in DCR space)	0x4 0010 0000	R/W/Clear (specific bits)	930
I2O0_ISEAT	I2O Slave Error Attribute Register (Also at address 0x006D in DCR space)	0x4 0010 0004	R	930
I2O0_ISEAD	I2O Slave Error Address Register (Also at address 0x006E in DCR space)	0x4 0010 0008	R	931
	Reserved.	0x4 0010 000C - 001C		n/a
I2O0_IDBEL	I2O Inbound Doorbell Register	0x4 0010 0020	R/W	931
	Reserved.	0x4 0010 0024 - 002F		n/a
I2O0_IHIS	I2O Host Interrupt Status	0x4 0010 0030	R	931
I2O0_IHIM	I2O Host Interrupt Mask	0x4 0010 0034	R	932
	Reserved.	0x4 0010 0038 -003F		n/a
I2O0_IHIQ	I2O Host Inbound Queue Port (32-bit version)	0x4 0010 0040	R/W	932
I2O0_IHOQ	I2O Host Outbound Queue Port (32-bit version)	0x4 0010 0044	R/W	933

Table 26-26. I2O Memory Mapped Register Summary (Continued)

Mnemonic	Register	Address	Access	Page
	Reserved.	0x4 0010 0048 - 004F		n/a
I2O0_IOPIS	I2O/DMA Interrupt Status	0x4 0010 0050	R	933
I2O0_IOPIM	I2O/DMA Interrupt Mask	0x4 0010 0054	R	934
I2O0_IOPIQ	I2O Inbound Queue Port (32-bit version)	0x4 0010 0058	R/W	935
I2O0_IOPOQ	Outbound Queue Port (32-bit version)	0x4 0010 005C	R/W	935
I2O0_IIFLH	I2O Inbound Free List Head Pointer	0x4 0010 0060	R/W	935
I2O0_IIFLT	I2O Inbound Free List Tail Pointer	0x4 0010 0062	R/W (specific bits)	936
I2O0_IIPLH	I2O Inbound Post List Head Pointer	0x4 0010 0064	R/W	936
I2O0_IIPLT	I2O Inbound Post List Tail Pointer	0x4 0010 0066	R/W (specific bits)	936
I2O0_IOFLH	I2O Outbound Free List Head Pointer	0x4 0010 0068	R/W	937
I2O0_IOFLT	I2O Outbound Free List Tail Pointer	0x4 0010 006A	R/W (specific bits)	937
I2O0_IOPLH	I2O Outbound Post List Head Pointer	0x4 0010 006C	R/W	937
I2O0_IOPLT	I2O Outbound Post List Tail Pointer	0x4 0010 006E	R/W (specific bits)	937
I2O0_IIDC	I2O Inbound Doorbell Clear Register	0x4 0010 0070	W	938
I2O0_ICTL	I2O Control	0x4 0010 0074	R/W	938
I2O0_IFCPP	I2O Free CDB Pointer Port (32-bit version)	0x4 0010 0078	W	938
	Reserved.	0x4 0010 007C - 007F		n/a
I2O0_MFAC0	MFA Size Count 0	0x4 0010 0080	R/W	939
I2O0_MFAC1	MFA Size Count 1	0x4 0010 0082	R/W	939
I2O0_MFAC2	MFA Size Count 2	0x4 0010 0084	R/W	939
I2O0_MFAC3	MFA Size Count 3	0x4 0010 0086	R/W	939
I2O0_MFAC4	MFA Size Count 4	0x4 0010 0088	R/W	939
I2O0_MFAC5	MFA Size Count 5	0x4 0010 008A	R/W	939
I2O0_MFAC6	MFA Size Count 6	0x4 0010 008C	R/W	939
I2O0_MFAC7	MFA Size Count 7	0x4 0010 008E	R/W	939
I2O0_IFCFH	I2O Free CDB FIFO Head Pointer	0x4 0010 0090	R/W	939
I2O0_IFCHT	I2O Free CDB FIFO Tail Pointer	0x4 0010 0092	R/W	940
	Reserved.	0x4 0010 0094 - 0097		n/a
I2O0_IIFMC	I2O Interrupt Frequency Mode Control	0x4 0010 0098	R/W	940
I2O0_IODB	I2O Outbound Doorbell	0x4 0010 009C	R/W	940
I2O0_IODBC	I2O Outbound Doorbell Clear	0x4 0010 00A0	W	941
I2O0_IFBAL	I2O FIFO Base Address Low	0x4 0010 00A4	R/W	941

Table 26-26. I2O Memory Mapped Register Summary (Continued)

Mnemonic	Register	Address	Access	Page
I2O0_IFBAH	I2O FIFO Base Address High	0x4 0010 00A8	R/W	942
I2O0_IFSIZ	I2O FIFO Size	0x4 0010 00AC	R/W	942
I2O0_ISPD0	I2O Scratchpad 0	0x4 0010 00B0	R/W	942
I2O0_ISPD1	I2O Scratchpad 1	0x4 0010 00B4	R/W	942
I2O0_ISPD2	I2O Scratchpad 2	0x4 0010 00B8	R/W	942
I2O0_ISPD3	I2O Scratchpad 3	0x4 0010 00BC	R/W	942
I2O0_IHIPL	I2O Host Inbound Queue Port Low (64-bit version)	0x4 0010 00C0	R/W	942
I2O0_IHIPH	I2O Host Inbound Queue Port High (64-bit version)	0x4 0010 00C4	R/W	943
I2O0_IHOPL	I2O Host Outbound Queue Port Low (64-bit version)	0x4 0010 00C8	R/W	943
I2O0_IHOPH	I2O Host Outbound Queue Port High (64-bit version)	0x4 0010 00CC	R/W	944
I2O0_IIIPL	I2O IOP Inbound Queue Port Low (64-bit version)	0x4 0010 00D0	R/W	944
I2O0_IIIPH	I2O IOP Inbound Queue Port High (64-bit version)	0x4 0010 00D4	R/W	944
I2O0_IIOPL	I2O IOP Outbound Queue Port Low (64-bit version)	0x4 0010 00D8	R/W	945
I2O0_IIOPH	I2O IOP Outbound Queue Port High (64-bit version)	0x4 0010 00DC	R/W	945
I2O0_IFCPL	I2O Free CDB Pointer Port Low (64-bit version)	0x4 0010 00E0	W	946
I2O0_IFCPH	I2O Free CDB Pointer Port High (64-bit version)	0x4 0010 00E4	W	946
	Reserved.	0x4 0010 00E8 - 00EC		n/a
I2O0_IOPT	I2O Observability Port (Also at offset 0x5F in DCR Space)	0x4 0010 00F0	R	946
	Reserved.	0x4 0010 00F4 - 00FF		n/a

26.6.3.1 I2O Status Register (I2O0_ISTS)

This register is used to report the status of the I2O Controller. All error conditions are logically ORed and forwarded to the 460EX/EXr/GT as one interrupt called I2O Error. This register is cleared on a system reset.

Figure 26-54. I2O Status Register (I2O0_ISTS)

31:11		Reserved.	
10	PMQA	PLB MIRQ Asserted: 0 Not asserted 1 Asserted	Writing a 1 to this bit will clear it
9:6	FACCS	I2O FIFO Access: 0000 Reset state 0001 Inbound Post List FIFO Write 1010 Inbound Post List FIFO Read 0011 Inbound Free List FIFO Write 0100 Inbound Free List FIFO Read 0101 Outbound Post List FIFO Write 0110 Outbound Post List FIFO Read 0111 Outbound Free List FIFO Write 1000 Outbound Free List FIFO Read 1001 Free CDB Pointer FIFO Read 1010 Free CDB Pointer FIFO Write	Defines the FIFO name and type of access that caused the error
5:3	STIPM	Status of the I2O PLB Master: 000 Normal Status 001 Wrong PLB Slave Size 010 PLB_MErr 011 PLB_MIRQ asserted while transfer in progress 100 PLB time-out asserted while waiting for address acknowledge. Note: Writing a 1 to any of these bits clears all three.	
2:0	STIPS	Status of the I2O PLB Slave: 000 Normal status 001 Read/Write to a reserved address or invalid byte enables 010 Write to a full FIFO 011 Read/Write to an I2O FIFO that does not exist. PLB MIRQ asserted while transfer in progress. Note: Writing a 1 to any of the three bits will clear all three bits and the slave error attribute and address registers.	

26.6.3.2 I2O Slave Error Attribute Register (I2O0_ISEAT)

The Master ID, Master Size, and BE bits of the Slave Error Attributes Register reflect attributes of the latest transfer the PLB Slave has address acknowledged on the PLB bus while the Error Transfer Type has normal status. When the PLB Slave detects an error, the register locks the attributes of the first error detected and remains locked. The Slave Error Attribute Register can be unlocked by clearing PLB Slave status bits of the I2O or DMA Status Registers.

Note: This address accesses a single Slave Error Attribute Register, visible from both I2O and DMA Register Interfaces.

Figure 26-55. I2O Slave Error Attribute Register (I2O0_ISEAT)			
31:25		Reserved.	
24:22	ERRTT	Error Transfer Type: 000 Normal Status 010 Read Transfer Error 011 Read Transfer Aborted 100 Write Transfer Error 101 Write Transfer Aborted	
21:18	MASID	Master ID of the PLB Transfer	
17:16	MASIZ	Size of the PLB Master	
15:0	BENB	Byte Enables for the Transfer	

26.6.3.3 I2O Slave Error Address Register (I2O0_ISEAD)

The Slave Error Address reflects the address of the latest transfer the PLB Slave has address acknowledged on the PLB bus while the Error Transfer Type bits of the Slave Error Attribute Register has normal status. When the PLB Slave detects an error, the register locks the address of the first error detected and remains locked. Clearing the PLB Slave status bits of the I2O or DMA Status Registers will unlock the Slave Error Address Register.

Note: This address accesses a single Slave Error Address Register, visible from both I2O and DMA Register Interfaces.

Figure 26-56. I2O Slave Error Address Register (I2O0_ISEAD)			
31:0	ISEAD	Slave Error Address	Contains the address associated with error accessing the I2O/DMA registers.

26.6.3.4 I2O Inbound Doorbell Register (I2O0_IDBEL)

When any bit in this register is set, an interrupt is generated to the 460EX/EXr/GT provided that the corresponding bit in the 460EX/EXr/GT Interrupt Mask Register is clear. Each bit is cleared by writing a 1 to the corresponding bit in the IOP Inbound Doorbell Clear Register.

Writes to this register must be 32-bit aligned writes.

Figure 26-57. I2O Inbound Doorbell Register (I2O0_IDBEL)			
31:0	IDBEL	Host Inbound Doorbell Register	Setting any bit in this register generates a interrupt in the 460EX/EXr/GT. Bits can only be cleared through the Outbound Doorbell Clear Register.

26.6.3.5 I2O Host Interrupt Status Register (I2O0_IHIS)

There are four sources for host interrupts: a General Doorbell signal, the Outbound PostList FIFO status, a Performant Mode Completion signal, and finally a general, multi-purpose interrupt. These interrupts are routed separately to the PCI interface so it can send separate message interrupts (if MSIs are supported).

Figure 26-58. I2O Host Interrupt Status Register (I2O0_IHIS)			
31	UNIRQ	Unassigned Interrupt	Set when the Unassigned Interrupt bit is set in the Outbound Doorbell register. It is cleared by writing a 1 to the Unassigned Interrupt bit in the Outbound Doorbell Clear register or through automatic clearing (if using MSIs).
30:4		Reserved.	
3	OPLNE	Outbound Post List FIFO Not Empty: 0 Outbound Post List FIFO is empty 1 Outbound Post List FIFO is not empty	This bit is set when the Outbound Post List FIFO is not empty. It is cleared when the Outbound Post List FIFO is empty.
2	GDIRQ	General Outbound Doorbell	Set if the General Doorbell Value bits have a non-zero value (located in the Outbound Doorbell register). This bit will be cleared only when the General Doorbell Value bits are ALL clear. This can be accomplished through a write to the Outbound Doorbell Clear register.
1		Reserved.	
0	PMIRQ	Performant Mode Completion	Set when the Performant Mode Completion bit is set in the Outbound Doorbell Register. It is cleared by writing a 1 to the Performant Mode Completion bit in the Outbound Doorbell Clear Register or through automatic clearing (if using MSIs).

26.6.3.6 I2O Host Interrupt Mask Register (I2O0_IHIM)

This register controls whether the host will actually receive interrupts generated from the Outbound Post List FIFO and the Outbound Doorbell registers.

Figure 26-59. I2O Host Interrupt Mask Register (I2O0_IHIM)			
31	UNIRQ	Unassigned Interrupt	When clear, the corresponding interrupt is routed to the host. When set, the interrupt is masked off.
30:4		Reserved.	
3	OPLNE	Outbound Post List FIFO Not Empty Mask	
2	GDIRQ	General Outbound Doorbell	
1		Reserved.	
0	PMIRQ	Performant Mode Completion	

26.6.3.7 I2O Host Inbound Queue Port, 32 bit-Version (I2O0_IHIQ)

This register allows the host or other IOPs to access the Inbound Post List FIFO or the Inbound Free List FIFO. On a read, the data returned is a Message Frame Address from the head of the Inbound Free List FIFO. The Inbound Free List Head Pointer is then incremented by eight bytes. Reads when the Inbound Free List FIFO is empty will return 0xFFFF_FFFF. In such a case, the Inbound Free List Head Pointer will not be incremented.

On a write, the data sent is interpreted as an MFA and placed at the tail of the Inbound Post List FIFO. The Inbound Post List Tail Pointer is then incremented by eight bytes. Writes of less than 32 bits are not supported. Writes when the Inbound Post List FIFO is full will be ignored, and the Inbound Post List Tail Pointer will not be incremented.

Figure 26-60. I2O Host Inbound Queue Port, 32-bit Version (I2O0_IHIQ)			
31:0	HIP32	Host Inbound Queue Port-32 Bit Version	On a read operation, this register contains the next Inbound Free List FIFO element. On a write operation, the data is taken as an MFA and placed in the Inbound Post List FIFO.

26.6.3.8 I2O Host Outbound Queue Port, 32-bit Version (I2O0_IHOQ)

This register allows the host or other IOPs to access the Outbound Post List FIFO or the Outbound Free List FIFO. On a read, the data returned is a Message Frame Address from the head of the Outbound Post List FIFO. The Outbound Post List Head Pointer is then incremented by eight bytes. Reads when the Outbound Post List FIFO is empty will return 0xFFFF_FFFF. In such a case, the Outbound Post List Head Pointer will not be incremented.

On a write, the data sent is interpreted as an MFA and placed at the tail of the Outbound Free List FIFO. The Outbound Free List Tail Pointer is then incremented by eight bytes. Writes of less than 32 bits are not supported. Writes when the Outbound Free List FIFO is full will be ignored, and the Outbound Free List Tail Pointer will not be incremented.

Figure 26-61. I2O Host Outbound Queue Port, 32-bit Version (I2O0_IHOQ)			
31:0	HOP32	Host Outbound Queue Port-32 Bit Version	On a read operation, this register contains the next Outbound Free List FIFO element. On a write operation, the data is taken as an MFA and placed in the Outbound Post List FIFO.

26.6.3.9 I2O/DMA Interrupt Status Register (I2O0_IOPIS)

The I2O/DMA unit is the source of several interrupt inputs of the 460EX/EXr/GT's Universal Interrupt Controller (UIC). Each bit in this register represents the source of an interrupt. That bit is set when the source requests service. These interrupts can be masked individually by means of the I2O/DMA Interrupt Mask (I2O0_IOPIM) register.

Figure 26-62. I2O/DMA Interrupt Status Register (I2O0_IOPIS)			
31:13		Reserved.	
12	IERR	I2O Error: 0 Normal operation 1 A bit in the I2O Status Register is set indicating I2O has detected an error	
11:9		Reserved.	
8	P1DE	DMA Error: 0 Normal 1 A bit in the DMA Status Register is set	
7	P1DCP	DMA Command Pointer FIFO Full: 0 The Pointer FIFO is not full 1 The Pointer FIFO is full	

6	P1DCA	DMA Command Status FIFO Attention: 0 The Command Status FIFO is not full and contains no interrupt-capable Command Status entries. 1 The Command Pointer Status FIFO is full or contains an interrupt-capable Command Status entry	
5:3		Reserved.	
2	IDSET	Inbound Doorbell Set: 0 Normal 1 A write operation has set a bit in the Inbound Doorbell Register	
1		Reserved.	
0	IPLNE	Inbound Post List FIFO Not Empty: 0 Inbound post list FIFO is empty 1 Inbound post List FIFO is Not Empty	

26.6.3.10 I2O/DMA Interrupt Mask (I2O0_IOPIM)

The Interrupt Mask Register corresponds to the I2O/DMA Interrupt Status Register (I2O0_IOPIS) and masks the interrupt source from generating an interrupt to the 460EX/EXr/GT's UIC. Each bit in this register corresponds with a bit in the I2O/DMA Interrupt Status Register; setting the mask register bit disables that source from causing an interrupt to the 460EX/EXr/GT. The value in the I2O/DMA Interrupt Mask Register does not affect the value in the I2O/DMA Interrupt Status Register. If a bit is cleared in the register, the corresponding interrupt status bit is enabled to drive the IOP interrupt signal.

Figure 26-63. I2O/DMA Interrupt Mask (I2O0_IOPIM)

31:13		Reserved.	
12	IERM	I2O Error Mask: 0 Allows I2O Error Interrupt to drive 460EX/EXr/GT's UIC 1 Masks I2O Error Interrupt	
11:9		Reserved.	
8	P1EM	DMA Error Mask: 0 Allows DMA Error Interrupt to drive 460EX/EXr/GT's UIC 1 Masks DMA Error Interrupt	
7	P1FFM	DMA Command Pointer FIFO Full Mask: 0 Allows DMA Command Pointer FIFO Full Interrupt to drive 460EX/EXr/GT's UIC 1 Masks DMA Command Pointer FIFO Full Interrupt	
6	P1SNE	DMA Command Status FIFO Not Empty Mask: 0 Allows DMA Command Status FIFO Not Empty (DMA Need Service) Interrupt to drive 460EX/EXr/GT's UIC 1 Masks DMA Command Status FIFO Not Empty Interrupt	
3:5		Reserved	
2	IDSM	Inbound Doorbell Set Mask: 0 Allows Inbound Doorbell Interrupt to drive 460EX/EXr/GT's UIC 1 Masks Inbound Doorbell Interrupt	
1		Reserved.	
0	IPNEM	I2O Inbound Post List FIFO Not Empty Mask: 0 Allows I2O Inbound Post List FIFO Not Empty Interrupt to drive 460EX/EXr/GT's UIC 1 Masks I2O Inbound Post List FIFO Not Empty Interrupt	

26.6.3.11 I2O Inbound Queue Port, 32-bit Version (I2O0_IOPIQ)

The 460EX/EXr/GT uses this port to access the Inbound Post List FIFO and the Inbound Free List FIFO. When written, a Message Frame Address (MFA) is placed at the tail of the Inbound Free List FIFO and the Inbound Free List Tail Pointer is incremented by eight bytes. A read retrieves an MFA from the head of the Inbound Post List FIFO and increments the Inbound Post List Head Pointer by eight bytes.

This port does not support writes of less than 32-bit writes. Only writes to this port that have all four PCI byte enables asserted will update the Inbound Free List FIFO and increment the corresponding Tail Pointer.

A write to this port when the Inbound Free List FIFO is full is ignored. This module responds normally, but will not update the Inbound Free List FIFO or corresponding Tail Pointer.

A read returns 0xFFFF_FFFF if the Inbound Post List FIFO is empty and the corresponding Head Pointer is not incremented.

Figure 26-64. I2O Inbound Queue Port, 32-bit Version (I2O0_IOPIQ)

31:0	IOPIQ	Inbound Queue Port-32-bit Version	On a read operation, this register contains the next Inbound Post List FIFO element. On a write operation, the data is taken as an MFA and placed at the tail of the Inbound Free List FIFO.

26.6.3.12 I2O Outbound Queue Port, 32-bit Version (I2O0_IOPOQ)

This port is used by the 460EX/EXr/GT to access the Outbound Post List FIFO and the Outbound Free List FIFO. When it is written, a Message Frame Address (MFA) is placed at the tail of the Outbound Post List FIFO and the Outbound Post List Tail Pointer is incremented by eight bytes. Reading this port retrieves and MFA from the head of the Outbound Free List FIFO and increments the Outbound Free List Head Pointer by eight bytes.

This port does not support writes less than 32 bits. Only writes having all four PCI byte enables asserted will update the Outbound Post List FIFO and increment the corresponding Tail Pointer.

A write to this port when the Outbound Post List FIFO is full is ignored. This module will respond normally, but will not update the Outbound Post List FIFO or the corresponding Tail Pointer.

Reads return 0xFFFF_FFFF if the Outbound Free List FIFO is empty. The corresponding Head Pointer is not incremented in this case

.

Figure 26-65. I2O Outbound Queue Port, 32-bit Version (I2O0_IOPOQ)

31:0	IOPOQ	Outbound Queue Port, 32-bit Version	On a read operation, this register contains the next Inbound Free List FIFO element. On a write operation, the data is taken as an MFA and placed at the tail of the Inbound Post List FIFO.

26.6.3.13 I2O Inbound Free List Head Pointer (I2O0_IIFLH)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-66. I2O Inbound Free List Head Pointer (I2O0_IIFLH)			
15		Reserved.	
14:0	IIFLH	I2O Inbound Free List Head Pointer	Head Pointer's current position in the Inbound Free List FIFO; this is a byte offset from the FIFO's base address.

26.6.3.14 I2O Inbound Free List Tail Pointer (I2O0_IIFLT)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-67. I2O Inbound Free List Tail Pointer (I2O0_IIFLT)			
15	FIFOF	FIFO Full: 0 Inbound Free List FIFO is not full 1 Inbound Free List FIFO is full	
14:0	IIFLT	Inbound Free List Tail Pointer	Tail Pointer's current position in the Inbound Free List FIFO; this is a byte offset from the FIFO's base address.

26.6.3.15 I2O Inbound Post List Head Pointer (I2O0_IIPLH)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-68. I2O Inbound Post List Head Pointer (I2O0_IIPLH)			
15		Reserved.	
14:0	IIPLH	I2O Inbound Post List Head Pointer	Head Pointer's current position in the Inbound Post List FIFO; this is a byte offset from the FIFO's base address.

26.6.3.16 I2O Inbound Post List Tail Pointer (I2O0_IIPLT)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-69. I2O Inbound Post List Tail Pointer (I2O0_IIPLT)			
15:14	FIFOF	FIFO Full: 0 Inbound Post List FIFO is not full 1 Inbound Post List FIFO is full	
13:0	IIPLT	Inbound Post List Tail Pointer	Tail Pointer's current position in the Inbound Post List FIFO; this is a byte offset from the FIFO's base address.

26.6.3.17 I2O Outbound Free List Head Pointer (I2O0_IOFLH)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-70. I2O Outbound Free List Head Pointer(I2O0_IOFLH)

15		Reserved.	
14:0	IOFLH	Outbound Free List Head Pointer	Head Pointer's current position in the Outbound Free List FIFO; this is a byte offset from the FIFO's base address.

26.6.3.18 I2O Outbound Free List Tail Pointer (I2O0_IOFLT)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-71. I2O Outbound Free List Tail Pointer (I2O0_IOFLT)

15	FIFOF	FIFO Full: 0 Outbound Free List FIFO is not full 1 Outbound Free List FIFO is full	
14:0	IOFLT	Outbound Free List Tail Pointer	Tail Pointer's current position in the Outbound Free List FIFO; this is a byte offset from the FIFO's base address.

26.6.3.19 I2O Outbound Post List Head Pointer (I2O0_IOPLH)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-72. I2O Outbound Post List Head Pointer(I2O0_IOPLH)

15		Reserved.	
14:0	IOPLH	I2O Outbound Post List Head Pointer	Head Pointer's current position in the Outbound Post List FIFO; a byte offset from the FIFO's base address.

26.6.3.20 I2O Outbound Post List Tail Pointer (I2O0_IOPLT)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-73. I2O Outbound Post List Tail Pointer (I2O0_IOPLT)

15	FIFOF	FIFO Full: 0 Outbound Post List FIFO is not full 1 Outbound Post List FIFO is full	

14:0	IOPLT	Outbound Post List Tail Pointer	Tail Pointer's current position in the Outbound Post List FIFO; this is a byte offset from the FIFO's base address.

26.6.3.21 I2O Inbound Doorbell Clear Register (I2O0_IIDC)

This register is used to clear any asserted bits in the Inbound Doorbell Register. Writing a 1 to any bit position of this register will clear the corresponding bit in the Inbound Doorbell Register.

Reads of this register will always return zero.

Figure 26-74. I2O Inbound Doorbell Clear Register (I2O0_IIDC)

31:0	IIDC	Inbound Doorbell Clear	When any bit in this register is set to 1, the corresponding bit in the Inbound Doorbell Register is also set to 0.

26.6.3.22 I2O Control Register (I2O0_ICTL)

Figure 26-75. I2O Control Register (I2O0_ICTL)

31:5		Reserved.	
4	PULP1	DMA Port Used for I2O Pull Transaction: 0 Reserved 1 Pull requests are handled by DMA	
3:2	PLBPR	I2O Control PLB Priority: 00 Lowest priority 01 Next lowest priority 10 Next lowest priority 11 Highest priority	
1	PULEN	I2O Pull Mode Enable 0 Pull disabled 1 Pull enabled	I2O0_ICTL reset value is 0x0000_0002
0		Reserved.	

26.6.3.23 I2O Free CDB Pointer Port, 32-bit Version (I2O0_IFCPP)

When this port is written, a Command Descriptor Block Pointer is placed at the tail of the Free Command Descriptor Block Pointer FIFO and the Free Command Descriptor Block Pointer FIFO Tail is incremented by eight bytes. The CDB Pointers pushed into the FIFO by means of this port are used exclusively by the I2O Controller logic.

Writes of less than 32 bits are not supported. Only writes that have all four PCI byte enables asserted will update the Free CDB FIFO and increment the corresponding Tail Pointer. A write to this port when the Free Command Descriptor Block FIFO is full will be ignored. This module will respond normally, but will not update the Free Command Descriptor Block Pointer FIFO or the corresponding Tail Pointer. The Free Command Descriptor Block Pointer Head Pointer is advanced only during an I2O Pull operation. The Command Descriptor Block Pointer FIFO Head Pointer advances when the DMA Engine has accepted the I2O request.

Figure 26-76. I2O Free CDB Pointer Port, 32-bit Version (I2O0_IFCPP)			
31:0	IFCPP	Free CDB Pointer Port	Any write to this register is interpreted as a Command Descriptor Block Pointer. The pointer is added at the tail of the Free Command Descriptor Block Pointer FIFO.

26.6.3.24 I2O MFA Size Count 0 - Count 7 (I2O0_MFAC0–I2O0_MFAC7)

These registers contain programmed transfer counts to use in I2O Pull operations. When an MFA is written to the Host Inbound Post List with bit 0 set, bits 3:1 determine which of the eight SG1 Count registers is used to specify the transfer count. Specifically:

MFA[3:1] = 000: Use MFA Size Count 0 register = I2O0_MFAC0

MFA[3:1] = 001: Use MFA Size Count 1 register = I2O0_MFAC1

MFA[3:1] = 010: Use MFA Size Count 2 register = I2O0_MFAC2

MFA[3:1] = 011: Use MFA Size Count 3 register = I2O0_MFAC3

MFA[3:1] = 100: Use MFA Size Count 4 register = I2O0_MFAC4

MFA[3:1] = 101: Use MFA Size Count 5 register = I2O0_MFAC5

MFA[3:1] = 110: Use MFA Size Count 6 register = I2O0_MFAC6

MFA[3:1] = 111: Use MFA Size Count 7 register = I2O0_MFAC7

The value contained in each register represents the number of 16-byte blocks to transfer. The default/reset value for each register is 2, giving a 32 byte Command Descriptor Block count.

The register to be used should be programmed ahead of time to ensure the entire message frame is transferred.

Figure 26-77. I2O MFA Size Count 0 - Count 7 (I2O0_MFAC0:I2O0_MFAC7)			
31:0	ISGCx	MFA Size Count X	The count stored in this register represents the number of 16-byte blocks to transfer in I2O pull operations.

26.6.3.25 I2O Free CDB FIFO Head Pointer (I2O0_IFCFH)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-78. I2O Free CDB FIFO Head Pointer (I2O0_IFCFH)			
15	FFUL	DMA Free CDB FIFO Full: 0 Free CDB FIFO is not full 1 Free CDB FIFO is full	
14:0	FCFTP	DMA Free CDB FIFO Head Pointer	Head Pointer's current position in the Free CDB FIFO; this is a byte offset from the FIFO's base address.

26.6.3.26 I2O Free CDB FIFO Tail Pointer (I2O0_IFCHT)

This pointer is automatically updated during normal operation. The user must ensure there is no access to the FIFO before directly writing this FIFO pointer. Otherwise, the FIFO may exhibit unpredictable behavior.

Figure 26-79. I2O Free CDB FIFO Tail Pointer (I2O0_IFCHT)

15	FIFOF	DMA Free CDB FIFO Full: 0 Free CDB FIFO is not full 1 Free CDB FIFO is full	
14:0	TPTR	DMA Free CDB FIFO Tail Pointer	Tail Pointer's current position in the Free CDB FIFO; this is a byte offset from the FIFO's base address.

26.6.3.27 I2O Interrupt Frequency Mode Control (I2O0_IIFMC)

This register provides a means to reduce a large number of interrupts from constantly being serviced. Interrupts are masked in one of two ways: leading edge and trailing edge.

In leading edge mode, any detected interrupt that goes active causes a delay counter to begin counting down to zero. The delay counter starts at the value specified by the Interrupt Delay Value field. While the delay counter is not zero, interrupts are masked. When the counter reaches zero, the interrupts are unmasked. Once all interrupts are serviced, the delay counter is reloaded and waits for the next interrupt to become active.

In trailing edge mode, the delay counter is started when all interrupts are detected as inactive. The delay counter starts at the value specified by the Interrupt Delay Value field. While the delay counter is not zero, any newly asserted interrupts are masked. When the delay counter reaches zero, any pending interrupts are unmasked. Once all interrupts are detected as inactive, the counter is reloaded, and begins counting down again.

To disable the interrupt delay feature, load the Interrupt Delay Value field with 00_0000h.

Figure 26-80. I2O Interrupt Frequency Mode Control (I2O0_IIFMC)

31	INTDM	Interrupt Delay Mode 0 Use trailing edge interrupt delay mode 1 Use leading edge interrupt delay mode	I2O0_IIFMC reset value is 0x8000_0000
30:24		Reserved.	
23:0	INTDV	Interrupt Delay Value	This is a delay counter to control the masking of interrupts

26.6.3.28 I2O Outbound Doorbell Register (I2O0_IODB)

When any bit in this register is set, an interrupt is generated to the Host provided the corresponding bit in the Outbound Doorbell Interrupt mask bit in the Host Interrupt Mask Register (I2O0_IHIM). Interrupt Mask Register is clear. Each bit is cleared by writing a 1 to the corresponding bit in the Outbound Doorbell Clear Register.

Writes to this register must be 32-bit aligned writes.

Figure 26-81. I2O Outbound Doorbell Register (I2O0_IODB)			
31	UNIRQ	Unassigned Interrupt	Setting this bit will send an interrupt to the host. This is a general purpose interrupt that will automatically clear itself if MSIs are being used. Otherwise, this bit can only be cleared through the Outbound Doorbell Clear register.
30:1	GDIRQ	General Outbound Doorbell	Setting any of these bits will generate a General Outbound Doorbell interrupt to the host, and is reflected in the Host Interrupt Status Register. These bits can only be cleared by writing a one to the corresponding location in the Outbound Doorbell Clear Register.
0	PMIRQ	Performant Mode Completion	Setting this bit will cause a Performant Mode Completion interrupt to the host. This bit can be cleared by writing a 1 to the corresponding bit of the Outbound Doorbell Clear Register, or through automatic clearing (if using MSIs).

26.6.3.29 I2O Outbound Doorbell Clear Register (I2O0_IODBC)

Figure 26-82. I2O Outbound Doorbell Clear Register (I2O0_IODBC)			
31	UNIRQ	Unassigned Interrupt	Writing a 1 to this bit will clear the Unassigned Interrupt bit in the Outbound Doorbell register.
30:1	GDIRQ	General Outbound Doorbell	Writing a 1 to any of these bits will clear the corresponding bit in the Outbound Doorbell register.
0	PMIRQ	Performant Mode Completion	Writing a 1 to this bit will clear the Performant Mode Completion bit in the Outbound Doorbell Register.

26.6.3.30 I2O FIFO Base Address Low Register (I2O0_IFBAL)

This register defines the starting memory location for the I2O and DMA FIFO memory space. The lower/least significant 12 bits of the address are reserved/always 0. Therefore, the starting address must be 4KB aligned.

Figure 26-83. I2O FIFO Base Address Low Register (I2O0_IFBAL)			
31:12	IFBAL	I2O FIFO Base Address Low	Most significant 20 bits of the starting address for the I2O and DMA FIFOs. The least significant 12 bits are always 0.
11:0		Reserved.	

26.6.3.31 I2O FIFO Base Address High Register (I2O0_IFBAH)

Figure 26-84. I2O FIFO Base Address High Register (I2O0_IFBAH)			
31:0	IFBAH	I2O FIFO Base Address High	The most significant 32 bits defining the base address for I2O/DMA FIFOs.

26.6.3.32 I2O FIFO Size Register (I2O0_IFSIZ)

Figure 26-85. I2O FIFO Size Register (I2O0_IFSIZ)			
31:29		Reserved.	
28	CDBFE	I2O Free CDB List FIFO Enable: 0 I2O Free CDB List FIFO does not exist 1 I2O Free CDB List FIFO exists	When bit = 1, I2O memory space is reserved for the corresponding I2O FIFO.
27:16	CDBFS	I2O Free CDB List FIFO Size	The value of this field plus one is the number of elements in the I2O Free CDB List FIFO(1-4096)
15	OPLFE	Outbound Post List FIFO Enable: 0 Outbound Post List FIFO does not exist 1 Outbound Post List FIFO exists	When bit = 1, I2O memory space is reserved for the corresponding I2O FIFO.
14	OFLFE	Outbound Free List FIFO Enable: 0 Outbound Free List FIFO does not exist 1 Outbound Free List FIFO exists	When bit = 1, I2O memory space is reserved for the corresponding I2O FIFO.
13	IPLFE	Inbound Post List FIFO Enable: 0 Inbound Post List FIFO does not exist 1 Inbound Post List FIFO exists	When bit = 1, I2O memory space is reserved for the corresponding I2O FIFO.
12	IFLFE	Inbound Free List FIFO Enable: 0 Inbound Free List FIFO does not exist 1 Inbound Free List FIFO exists	When bit = 1, I2O memory space is reserved for the corresponding I2O FIFO.
11:0	IFSIZ	I2O FIFO Size	The value of this field plus one is the number of supported MFAs (1–4096)

26.6.3.33 I2O Scratch Pad 0–3 (I2O0_ISPD0–I2O0_ISPD3)

The following figure defines bits in register I2O0_ISPDn which represents the four identical scratch pad registers, I2O0_ISPD0 through I2O0_ISPD3.

Figure 26-86. I2O Scratch Pad 0–3 (I2O0_ISPD0–I2O0_ISPD3)			
31:0	ISPDx	Scratch pad used by firmware during the boot operation. This register is accessible by means of I/O cycles.	Where n represents the scratch pad numbers 0 through 3.

26.6.3.34 I2O Host Inbound Queue Port Low, 64-bit Version (I2O0_IHIPL)

The Host Inbound Post List FIFO and the Host Inbound Free List FIFO are always managed as 64-bit FIFOs. This register can be accessed by means of a single 64-bit operation or by two successive 32-bit operations.

For 64-bit access, simply write to this register with the appropriate address and byte enable combination.

To write the full 64 bits using 32-bit access, the upper 32 bits should be written first, followed by the lower 32 bits. The value written to the upper 32 bits is saved until the lower 32 bits (at offset 0x40 or 0xC0) are written. Writing to the lower 32 bits pushes the entire 64-bit MFA value onto the Host Inbound Post List FIFO, and the saved upper 32 bits are returned to zeroes. If the lower 32 bits are written (at offset 0x40 or 0xC0) without writing the upper 32 bits first, then the upper 32 bits will contain zeros. Note that the value written to the upper 32 bits will be cleared only after a write to the lower 32 bits.

Figure 26-87. I2O Host Inbound Queue Port Low, 64-bit Version (I2O0_IHIPL)

31:0	IHIPL	Host Inbound Queue Port Low, 64-bit Version	This port is equivalent to the Host Inbound Queue Port (32-bit Version) at offset 0x40.

26.6.3.35 I2O Host Inbound Queue Port High, 64-bit Version (I2O0_IHIPH)

This port is used by the Host or other IOPs to access the upper 32 bits of the Inbound Post List FIFO and the Inbound Free List FIFO. Writing this register stores the data as the upper 32 bits of a 64-bit MFA. When the lower 32 bits are written (through either port 0x40 or 0xC0), the concatenated 64-bit value is a Message Frame Address placed in the Inbound Post List FIFO. In addition, the upper 32 bits sent to this register are cleared to zeroes once the lower 32 bits are written. On a read, this port gives the upper 32 bits of an MFA from the head of the Inbound Free List FIFO.

This port does not support writes of less that 32 bits. Only writes to this port that have all four PCI byte enables asserted will be saved.

Figure 26-88. I2O Host Inbound Queue Port High, 64-bit Version (I2O0_IHIPH)

31:0	IHIPH	Host Inbound Queue Port High, 64-bit Version	On a read operation, this register returns the upper 32 bits of the next Inbound Free List FIFO element. On a write operation, the data is saved as the upper 32 bits of a 64-bit MFA to be placed in the Inbound Post List FIFO.

26.6.3.36 I2O Host Outbound Queue Port Low, 64-bit Version (I2O0_IHOPL)

The Host Outbound Free List FIFO and the Host Outbound Post List FIFO are always managed as 64-bit FIFOs. This register can be accessed by means of a single 64-bit operation or by two successive 32-bit operations.

For 64-bit access, simply write to this register with the appropriate address and byte enable combination.

To write the full 64 bits of the Host Outbound Queue Port, the upper 32 bits should be written first, followed by the lower 32 bits. The value written to the upper 32 bits is saved until the lower 32 bits (at offset 0x44 or 0xC8) are written. Writing to the lower 32 bits pushes the entire 64-bit MFA value onto the Host Outbound Free List FIFO, and the saved upper 32 bits are returned to zeroes. If the lower 32 bits are written (at offset 44h or C8h) without writing the upper 32 bits first, then the upper 32 bits will contain zeroes. Note that the value written to the upper 32 bits will be cleared only after a write to the lower 32 bits.

Figure 26-89. I2O Host Outbound Queue Port Low, 64-bit Version (I2O0_IHOPL)

31:0	IHOPL	Host Outbound Queue Port Low, 64-bit Version	This port is equivalent to the Host Outbound Queue Port (32-bit Version) at offset 0x44.

26.6.3.37 I2O Host Outbound Queue Port High, 64-bit Version (I2O0_IHOPH)

This port is used by the Host to access the upper 32 bits of the Outbound Free List FIFO and the Outbound Post List FIFO. Writing this register stores the data as the upper 32 bits of a 64-bit MFA. When the lower 32 bits are written (through either port 0x44 or 0xC8), the concatenated 64-bit value is a Message Frame Address placed in the Outbound Free List FIFO. In addition, the upper 32 bits sent to this register are cleared to zeroes once the lower 32 bits are written. On a read, this port gives the upper 32 bits of an MFA from the head of the Outbound Post List FIFO.

This port does not support writes of less that 32 bits. Only writes to this port that have all four PCI byte enables asserted will be saved.

A read will return 0xFFFF_FFFF if the Outbound Post List FIFO is empty.

Figure 26-90. I2O Host Outbound Queue Port High, 64-bit Version (I2O0_IHOPH)

31:0	IHOPH	Host Outbound Queue Port High, 64-bit Version	On a read operation, this register returns the upper 32 bits of the next Outbound Post List FIFO element. On a write operation, the data is saved as the upper 32 bits of a 64-bit MFA to be placed in the Outbound Free List FIFO.

26.6.3.38 I2O Inbound Queue Port Low, 64-bit Version (I2O0_IIIPL)

The 460EX/EXr/GT's Inbound Post List FIFO and the 460EX/EXr/GT's Inbound Free List FIFO are always managed as 64-bit FIFOs. This register can be accessed by means of a single 64-bit operation or by two successive 32-bit operations.

For 64-bit access, simply write to this register with the appropriate address and byte enable combination.

To write the full 64 bits of the 460EX/EXr/GT's Inbound Queue Port, the upper 32 bits should be written first, followed by the lower 32 bits. The value written to the upper 32 bits is saved until the lower 32 bits (at offset 0x58 or 0xD0) are written. Writing to the lower 32 bits pushes the entire 64-bit MFA value onto the 460EX/EXr/GT's Inbound Free List FIFO, and the saved upper 32 bits are returned to zeroes. If the lower 32 bits are written (at offset 0x58 or 0xD0) without writing the upper 32 bits first, then the upper 32 bits will contain zeroes. Note that the value written to the upper 32 bits will be cleared only after a write to the lower 32 bits.

Figure 26-91. I2O Inbound Queue Port Low, 64-bit Version (I2O0_IIIPL)

31:0	IIIPL	Inbound Queue Port Low, 64-bit Version	This port is equivalent to the IOP Inbound Queue Port (32-bit Version) at offset 0x58.

26.6.3.39 I2O Inbound Queue Port HIgh, 64-bit Version (I2O0_IIIPH)

This port is used by the 460EX/EXr/GT to access the upper 32 bits of the Inbound Post List FIFO and the Inbound Free List FIFO. Writing this register stores the data as the upper 32 bits of a 64-bit MFA. When the lower 32 bits are written (through either port 0x58 or 0xD0), the concatenated 64-bit value is a Message Frame Address placed in the Inbound Free List FIFO. In addition, the upper 32 bits sent to this register are cleared to zeroes once the lower 32 bits are written. On a read, this port gives the upper 32 bits of an MFA from the head of the Inbound Post List FIFO.

This port does not support writes of less that 32 bits. Only writes to this port that have all four PCI byte enables asserted will be saved.

A read will return 0xFFFF_FFFF if the Inbound Post List FIFO is empty.

Figure 26-92. I2O Inbound Queue Port High, 64-bit Version (I2O0_IIIPH)			
31:0	IIIPH	Inbound Queue Port High, 63-bit Version	On a read operation, this register returns the upper 32 bits of the next Inbound Post List FIFO element. On a write operation, the data is saved as the upper 32 bits of a 64-bit MFA to be placed in the Inbound Free List FIFO.

26.6.3.40 I2O Outbound Queue Port Low, 64-bit Version (I2O0_IIOPL)

The Outbound Post List FIFO and the Outbound Free List FIFO are always managed as 64-bit FIFOs. This register can be accessed by means of a single 64-bit operation or by two successive 32-bit operations.

For 64-bit access, simply write to this register with the appropriate address and byte enable combination.

To write the full 64 bits of the IOP Outbound Queue Port, the upper 32 bits should be written first, followed by the lower 32 bits. The value written to the upper 32 bits is saved until the lower 32 bits (at offset 0x5C or 0xD8) are written. Writing to the lower 32 bits pushes the entire 64-bit MFA value onto the Outbound Post List FIFO, and the saved upper 32 bits are returned to zeroes. If the lower 32 bits are written (at offset 0x5C or 0xD8) without writing the upper 32 bits first, then the upper 32 bits will contain zeroes. Note that the value written to the upper 32 bits will be cleared only after a write to the lower 32 bits.

Figure 26-93. I2O Outbound Queue Port Low, 64-bit Version (I2O0_IIOPL)			
31:0	IIOPL	Outbound Queue Port Low, 64-bit Version	This port is equivalent to the Outbound Queue Port (32-bit Version) at offset 0x5C.

26.6.3.41 I2O Outbound Queue Port High, 64-bit Version (I2O0_IIOPH)

This port is used by the 460EX/EXr/GT to access the upper 32 bits of the Outbound Post List FIFO and the Outbound Free List FIFO. Writing this register stores the data as the upper 32 bits of a 64-bit MFA. When the lower 32 bits are written (through either port 0x5C or 0xD8), the concatenated 64-bit value is a Message Frame Address placed in the Outbound Post List FIFO. In addition, the upper 32 bits sent to this register are cleared to zeroes once the lower 32 bits are written. On a read, this port gives the upper 32 bits of an MFA from the head of the Outbound Free List FIFO.

This port does not support writes of less that 32 bits. Only writes to this port that have all four PCI byte enables asserted will be saved.

Reads return 0xFFFF_FFFF if the Outbound Free List FIFO is empty.

Figure 26-94. I2O Outbound Queue Port High, 64-bit Version (I2O0_IIOPH)			
31:0	IIOPH	Outbound Queue Port High, 64-bit Version	On a read operation, this register returns the upper 32 bits of the next Outbound Free List FIFO element. On a write operation, the data is saved as the upper 32 bits of a 64-bit MFA to be placed in the Outbound Post List FIFO.

26.6.3.42 I2O Free CDB Pointer Port Low, 64-bit Version (I2O0_IFCPL)

The Free CDB Pointer FIFO is always managed as a 64-bit FIFO. This register can be accessed by means of a single 64-bit operation or by two successive 32-bit operations.

For 64-bit access, simply write to this register with the appropriate address and byte enable combination.

To write the full 64 bits of the Free CDB Pointer FIFO, the upper 32 bits should be written first, followed by the lower 32 bits. The value written to the upper 32 bits is saved until the lower 32 bits (at offset 0x78 and 0xE0) are written. Writing to the lower 32 bits pushes the entire 64-bit CDB Pointer into the Free CDB Pointer FIFO, and the saved upper 32 bits are returned to zeroes. If the lower 32 bits are written (at offset 0x78 or 0xE0) without writing to the upper 32 bits first, then the upper 32 bits will contain zeroes. Note that the value written to the upper 32 bits will be cleared only after there is a write to the lower 32 bits.

Figure 26-95. I2O Free CDB Pointer Port Low, 64-bit Version (I2O0_IFCPL)

31:0	IFCPL	Free CDB Pointer Port Low, 64-bit Version	This port is equivalent to the Command Descriptor Block Command Pointer Port (32-bit Version) at offset 0x78

26.6.3.43 I2O Free CDB Pointer Port High, 64-bit Version (I2O0_IFCPH)

When this port is written, the upper 32 bits of a COmmand Descriptor Block Pointer is saved (waiting for the lower 32 bits to be written). The CDB Pointers pushed into the FIFO by means of this port are used exclusively by the I2O Controller logic. This port does not support writes of less than 32 bits. Only writes to this port that have all four PCI byte enables asserted will be saved.

The Free Command Descriptor Block Pointer FIFO Head Pointer is advanced only during an I2O Pull operation.

Figure 26-96. I2O Free CDB Pointer Port High, 64-bit Version (I2O0_IFCPH)

31:0	IFCPH	I2O Free CDB Pointer Port, High, 64-bit Version	On a write operation, the data is saved as the upper 32 bits of a Command Descriptor Block Pointer FIFO when the lower 32 bits are written.

26.6.3.44 I2O Observability Port Register (I2O0_IOPT)

Figure 26-97. I2O Observability Port Register (I2O0_IOPT)

31:22	RRVEC	I2O Register Request Vector	
21	PULSL	I2O Pull Select	
20:11		Reserved.	
10:8	PSSM	PLB Slave State Machine	
7:6	MPMPS	FIFO Manager PLB Master Port Select	
5:3	MPMSM	FIFO Manager PLB Master State Machine State	
2:0	BESMS	I2O Register -- (Back End) State Machine State	

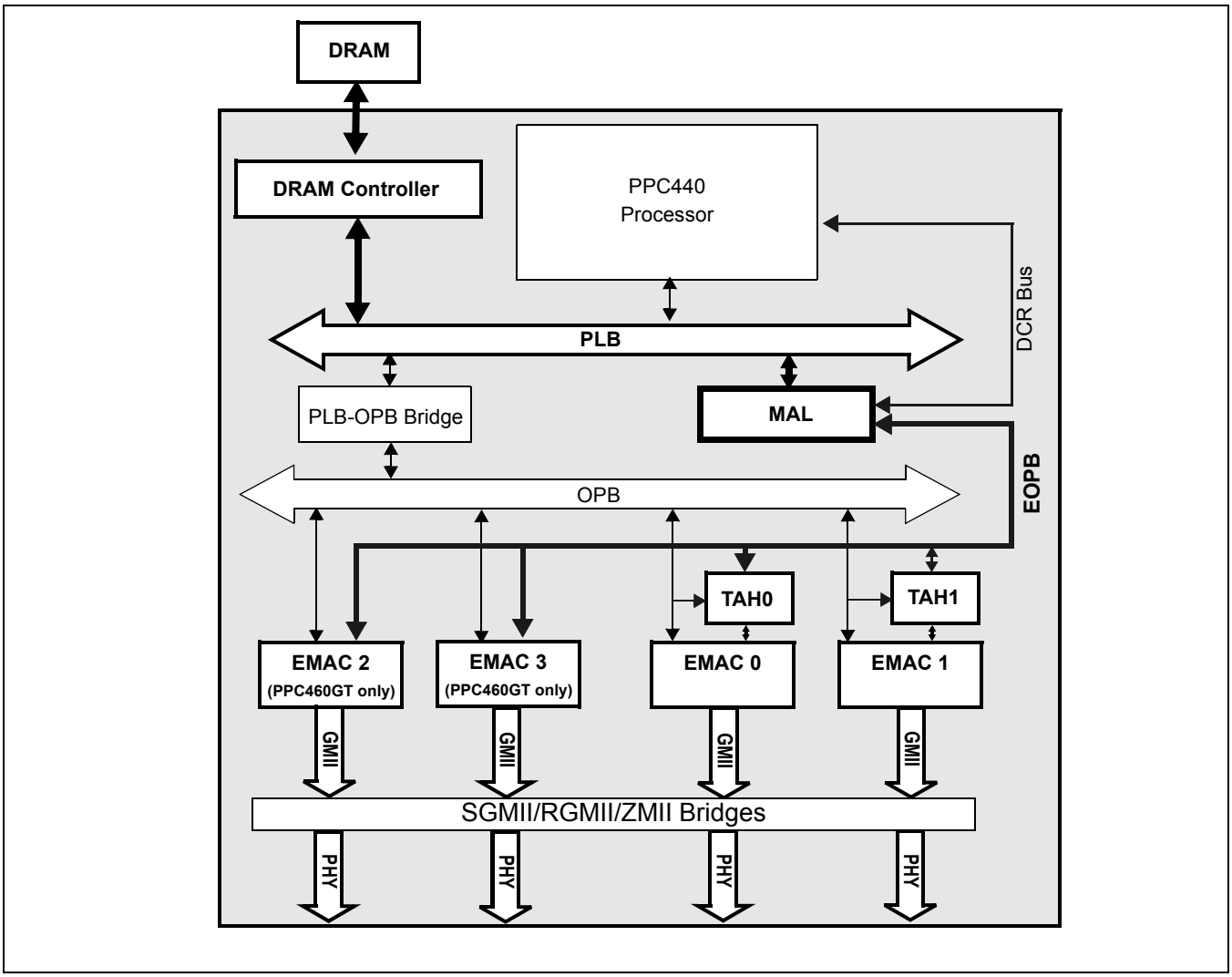
27. Memory Access Layer

The Memory Access Layer (MAL) provides specialized Direct Memory Access (DMA) data transfer service for one or more Ethernet Media Access Controller (EMAC) cores. It is dedicated to performing data transfers between the EMACs, and any PLB-accessible memory, commonly DRAM.

To interface with the EMACs, MAL utilizes a specialized dedicated On-chip Peripheral Bus (OPB), identified as the Extended OPB (EOPB). EOPB has additional side-band signals beyond the standard OPB to enable EMACs to request service and signal completion of transfers. MAL is both a PLB master and an EOPB master, meaning it initiates transfers on both of these buses.

Figure 27-1 Illustrates the MAL interfaces to PLB and EMACs provided on the PPC460EX/EXr/GT. Bolded busses and cores are involved with data flow between the Ethernet PHY and Ethernet frame buffers located, typically in DRAM. After an EMAC wins arbitration for the MAL to perform a transfer request, the MAL transfers data between DRAM, or any PLB accessible memory, and the EMAC.

Figure 27-1. General PPC460EX/EXr/GT Structure



The productnumber4 has four (4) Ethernet-MACs (EMACs) and the 460EX/EXr has two (2) EMACs. All are serviced by a single Memory Access Layer (MAL) core, identified as MAL0 (MAL #0).

The MAL has multiple channels to provide service to the EMACs concurrently. There are two types of channels; transmit (TX) for transmitting egress data to the EMAC, and receive (RX) for receiving ingress data from the EMAC. A pair of channels is used per EMAC, one physical RX channel, and one TX channel. Transmit and receive transfers can be performed simultaneously by the MAL since the EOPB has separate read and write data buses.

Each physical RX channel from the EMAC supports up to eight virtual RX channels. Virtual channels can be used as part of a system level solution for supporting Ethernet quality of service (QoS) (IEEE 802.1p Priority queueing). The virtual channel number is selected by the User Priority bits 7:5 of the first Octet of a VLAN tag. See *Section 28.9.2 VLAN Tagged Packet Reception*.

Ethernet-frame data to be transferred by the MAL to/from the EMAC are stored in buffers located in memory, typically DRAM. Frames are composed of one or more buffers. The MAL utilizes buffers by processing buffer-descriptors (BD's), which are stored in a BD-table, and used sequentially in the order stored in the table. Each TX channel and each virtual RX channel has its own BD table.

Each BD contains an address pointing to a buffer, and a dual purpose control and completion-status word. The control-word configures the EMAC before each data transfer, and the completion-status word is read from the EMAC and placed into the buffer-descriptor at the end of data transfer to/from the buffer. Device-driver software configure the BDs to provide the MAL with the buffers address, the empty or used state of the buffer, and the EMACs' operational mode. Upon completion of the transfer, which can optionally cause an IRQ, device-driver software uses the completion-status word stored in the buffer-descriptor to examine the data-transfer completion-status.

The MAL transfers data to/from the buffers as controlled by the contents of the BDs. Typically, to request transmission of a network packet/datagram by the Ethernet network data-link layer, Ethernet client software, such as a networking protocol stack, calls an Ethernet device-driver which controls both the EMAC and the MAL cores. This Ethernet driver is also responsible to segment the packet/datagram into Ethernet frames, and the frames into one or more data buffers.

27.1 MAL Features and Organization

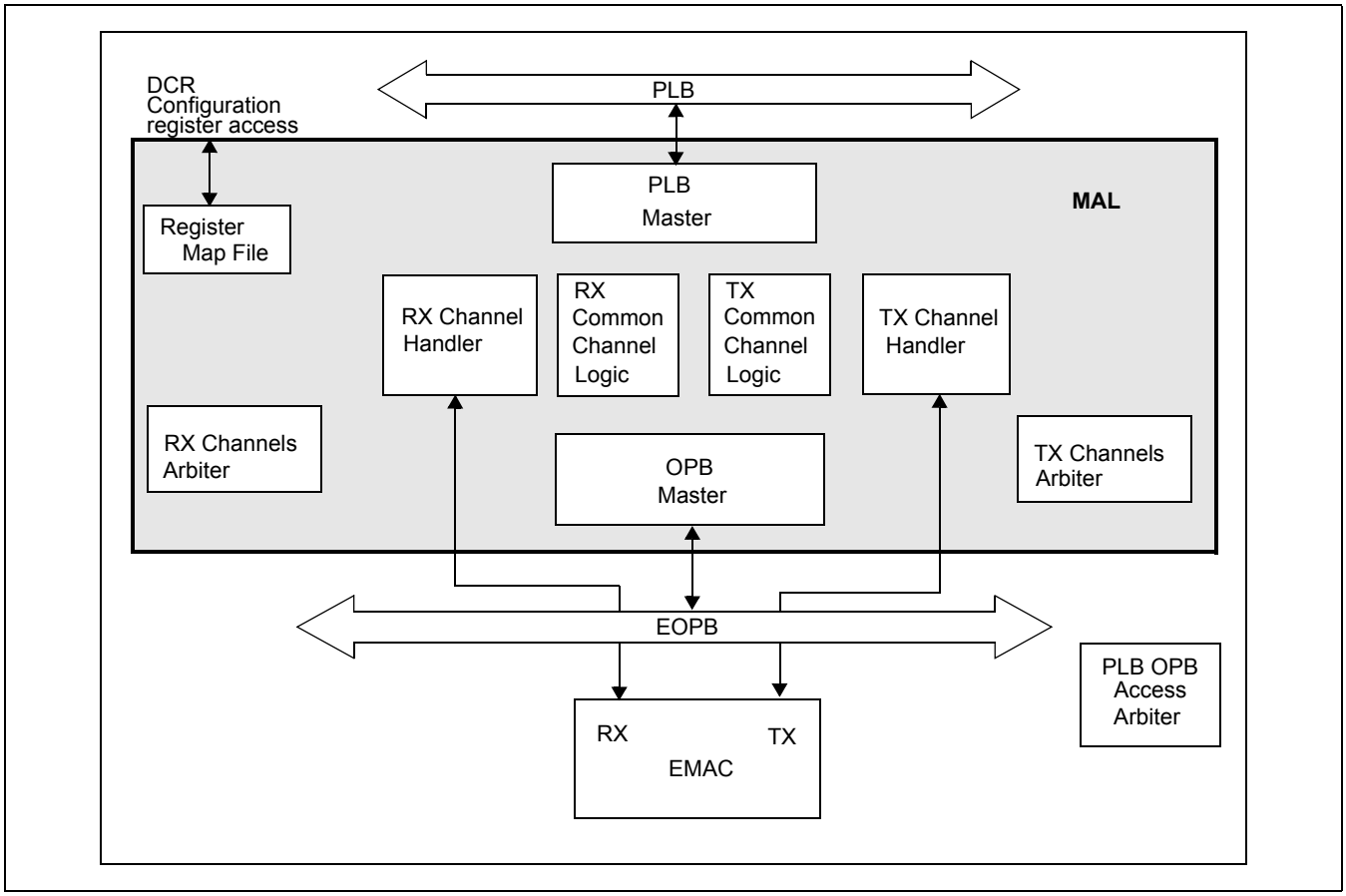
- Provides a TX and a RX service request channel for each EMAC core. 4 TX and 4 Physical RX are utilized by 460GT or 2 TX and 2 Physical RX by 460EX/EXr.
- Eight virtual RX channels for each physical RX channel. Virtual RX channels assist software in supporting Ethernet quality of service (QoS) (IEEE 802.1p Priority queueing).
- Each channel uses a separate buffer-descriptor table. Up to 256 buffer-descriptors in each buffer-descriptor table supporting up to 256 buffers per channel.
- Supports use of multiple buffers per Ethernet frame.
- No restrictions on buffer starting address alignment in memory. By using transfer size-aligned buffers burst operation with off-chip memories such as DRAM is enabled.
- Aligned bus accesses to enable burst operation with external memories
- Configurable receive buffer size, on a per channel basis.
- No minimum TX buffer size, does not have to be at least 64 bytes to meet Ethernet minimum frame size specifications if automatic padding of TX frames is enabled.
- Maximum buffer sizes of 4095 bytes (TX) and 4080 bytes (RX)
- Up to 256 descriptors in the buffer descriptor table per channel
- Configures the EMAC on each transfer, as specified in the buffer-descriptor control/completion-status word field.

- Updates the buffer-descriptor control/completion-status field at the end of each frames' transfer, according to the status as read from the EMAC.
- Per-buffer or per-frame interrupt, maskable per channel.
- Concurrent operation of RX and TX channels with RX channels prioritized above TX channels.
- Configured via Device Control Registers (DCRs) using **mtdcr** and **mfdcr** instructions.
- Configurable PLB service-request arbitration-priority
- PLB and OPB error reporting, via status registers.

27.1.1 MAL Internal Structure

Figure 27-2 illustrates the MAL functional blocks.

Figure 27-2. MAL Internal Structure



27.1.1.1 PLB Master

The PLB Master interface logic performs PLB transactions for the MAL, and is used to transfer data between a EMAC and memory, fetch buffer-descriptors, and write status regarding data transfer completion to the BD.

27.1.1.2 EOPB Master

The EOPB master logic performs EOPB transactions for the MAL, and is used to transfer data between an EMAC and memory.

27.1.1.3 TX Channel Handler

The TX channel handler is a dedicated section for each TX channel. It keeps a record of the descriptor information and the current state of each channel.

27.1.1.4 RX Channel Handler

The RX channel handler has dedicated registers for each RX channel. It keeps a record of the buffer-descriptor information and the current state of each channel.

27.1.1.5 TX Channel Arbiter

The TX channel arbiter, connected to all service-request signals from each TX channel, arbitrates between the TX channels and decides which TX channel gains access to the TX-common channel-logic.

27.1.1.6 RX Channel Arbiter

The RX channel arbiter, connected to all service-request signals from each RX channel, arbitrates between the RX channel service-requests, and decides which RX channel gains access to the RX-common channel-logic.

27.1.1.7 TX Common Channel-Logic

The TX-common channel-logic is shared by all TX channels. It services a single TX channel transfer request at a time as selected by the TX arbiter logic. This logic activates the PLB and OPB masters for data-transfer and buffer-descriptor read and write transactions.

27.1.1.8 RX Common Channel-Logic

The RX-common channel-logic is shared by all RX channels. It services a single RX channel at a time as selected by the RX arbiter logic. This logic activates the PLB and OPB masters for data-transfer and buffer-descriptor read and write transactions.

27.1.1.9 Register Map File

The register-file is used to contain the MAL configuration registers and to read its status registers. Software accesses the MAL register-file using the **mtdcr** and **mfdcr** instructions.

27.2 MAL0 Interfaces and Channel Assignments

A MAL is comprised of 32 channels where two channels (TX and RX pair) are needed for each EMAC. The 460EX/EXr supports four channels, two transmit and two receive and the productnumber4 supports eight channels, four transmit and four receive. Unused MAL channels are not connected to any other logic and require no configuration or servicing by software. See *Table 27-1* for MAL0 service-channel assignments.

In the PPC460EX/EXr/GT, MAL uses a PLB to EOPB clock frequency ratio of 1:1, i.e. the EOPB interface between MAL0 and the EMAC cores is clocked at the same frequency as the PLB interface between MAL0 and the PLB.

Table 27-1. MAL0 Channel Assignment

MAL0 Channel	EMAC Channel
Physical RX Channel 0 (Virtual Channel 0-7) For non- VLAN frames, Virtual Channel 0 is the default.	EMAC0 RX Channel
Physical RX Channel 1 (Virtual Channel 8-15) For non- VLAN frames, Virtual Channel 8 is the default.	EMAC1 RX Channel
Physical RX Channel 2 (PPC460GT only) (Virtual Channel 16-23) For non- VLAN frames, Virtual Channel 16 is the default.	EMAC2 RX Channel
Physical RX Channel 3 (PPC460GT only) (Virtual Channel 24-31) For non- VLAN frames, Virtual Channel 24 is the default.	EMAC3 RX Channel
TX Channel 0	EMAC0 TX Channel
TX Channel 1	EMAC1 TX Channel
TX Channel 2 (PPC460GT only)	EMAC2 TX Channel
TX Channel 3 (PPC460GT only)	EMAC3 TX Channel

27.3 MAL Operations

27.3.1 Buffers and Buffer-Descriptors (BD's)

Each transmit and virtual receive MAL service-request channel has a independent buffer-descriptor (BD) table-pointer register which points to an address in memory that contains a circular-list of BDs that together form a buffer-descriptor table (BDT). Each individual transmit or receive channel has its own BDT.

Each BD in a BDT has a pointer to a buffers' address in memory, an EMAC control word, and a transfer-completion status-word. Software controls buffer usage by configuring BDs, and MAL registers.

Note: The physical memory used for buffers and BDTs can be located anywhere in the PLB address-space. However, all buffers and BDs must be contained within the same 32 bit address space. Crossing a 4GB address boundary is not supported.

27.3.2 MAL Usage of Buffers and Buffer-Descriptors

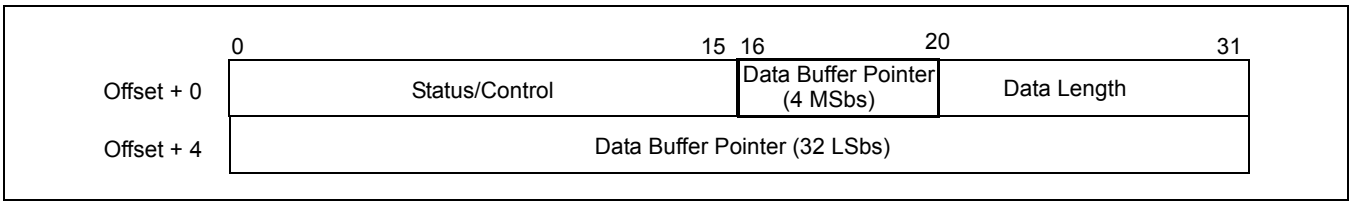
When transmitting frames, the device-driver/network-protocol stack software places frame-data into buffers and when receiving frames, software provides empty buffers. Concurrently, the MAL processes the BDs, transfers the frame-data stored in the buffers to/from the EMAC, and updates the completion-status fields of the utilized BDs.

MAL has a Channel Table Pointer Register for each of its channels which points to the channels BDT. The EMAC (EMAC in the PPC460EX/EXr/GT) device-driver initialization software sets the channel-table pointer registers to point to the starting address of the BDT for each utilized channel.

The BDT is a circular-queue list with a programmable length. The last BD in the table is defined by setting the Wrap bit in its status/control field (see "Status/Control Field Format" on page 961). If there is no Wrap bit set in the table, then the MAL automatically wraps after processing 256 BDs, which is the maximum number of BDs supported per channel.

The format of the BD as shown in *Figure 27-3* is the same for all EMACs, and has the same structure for both transmit and receive. The most significant 16-bit halfword in each BD contains a status/control halfword. This field has two parts: the first part (6 bits) is BD handling information used by the MAL for BD processing, the second field (10 bits) is content specific for each EMAC. The second halfword determines the 4 MSbs of the data buffer pointer and the data length (in bytes) referenced in this BD. The second word in the BD contains the 32 LSbs of the data buffer pointer that points to the actual data buffer in memory. It is suggested that each data buffer start on a cache line boundary and be a multiple of a cache line in size if it resides in cachable memory. (The cache line size in the PPC460EX/EXr/GT processor core is 32 bytes.)

Figure 27-3. Buffer Descriptor (BD) Structure



A frame may reside in as many buffers as necessary (transmit or receive); therefore, multiple BDs will be accessed as required to process the entire Ethernet frame, A buffer can have a maximum length of 2^{12} -16 bytes, i.e. 4096 – 16 = 4,080 bytes.

For TX channels, the *data-length* field defines the number of bytes in the data buffer that are to be transferred to the EMAC. The *data-length* field is programmed by the Ethernet device driver software when the data to be transmitted is placed into the data buffer,

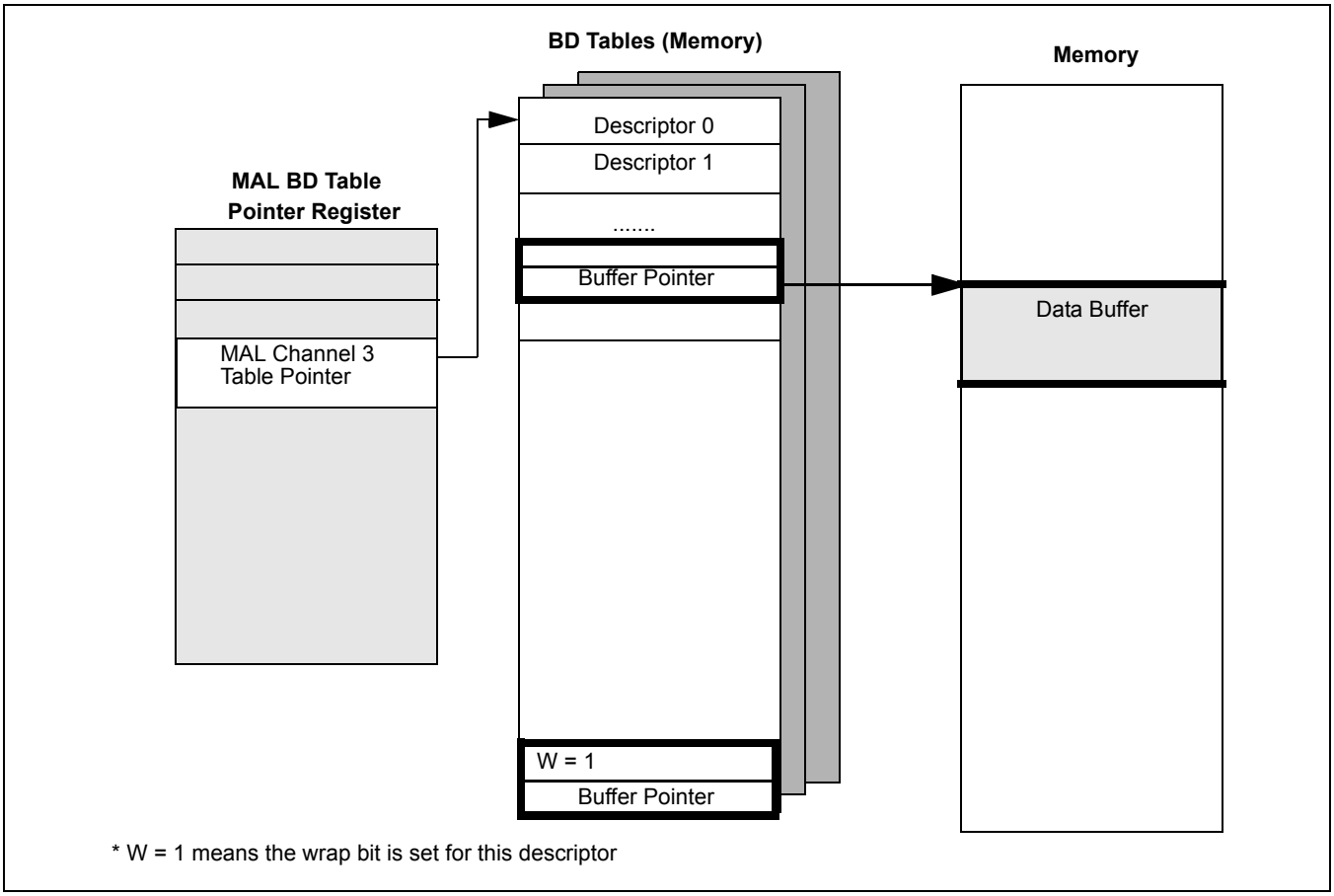
For RX channels, the *data-length* field contains the number of bytes placed into the data-buffer by the MAL. This field is written by MAL when it finishes using a data-buffer. See *Receive Software Interface* on page 959 for more information.

When processing a frame, MAL does not require that all buffers to be used for the frame are already Ready or Empty. It is only required that the buffers be ready by the time MAL is processing them. Failure of the software to provide the BDs in time results in an Descriptor Error.

The rate that MAL processes buffers is determined by many factors including the physical-media speed, MAL, DRAM and PLB utilization, and PLB clock rate. MAL utilization is controlled by the activity level of the other EMACS usage of the MAL TX and RX channels because the MAL simultaneously services only one TX and one RX channel at a time due to the EOPB being a single instance of a full-duplex OPB. The minimum time to process a buffer = # of bytes in buffer/(PLB-clock-rate * PLB-bus-width in bytes), without accounting for any overhead on the EOPB, PLB or DRAM busses. For example, the PLB 128 bit data width @ 166 MHz clock rate, moves 2.666 GB/s assuming no contention for MAL, PLB or DRAM resources. Therefore, a 1000 byte buffer takes the MAL at least 375 nS to process.

Figure 27-4 describes the MAL data structures used to hold an Ethernet frame in memory.

Figure 27-4. Packet Memory Structure



27.3.3 Buffer-Descriptors and Cache-Coherency

During operation, MAL modifies the contents of BDs with completion-status results read from the EMAC, directly without any indication to the processor core, or its L1 cache. There is no hardware enforced cache-coherency with regard to MAL writes. Also, multiple BDs fit in a single cache-line of data, and so neighboring BDs may be modified by MAL without being visible to software. A subsequent cache-flush would overwrite valid BD data with stale data from the cached line. Therefore Data-cache coherency must be ensured by design of the software and to reduce device-driver software requirements with regards to managing BD cache-coherency, the BD tables should ***only*** be placed in non-cached memory, and this is very highly-recommend for reliable operation.

If use of cache-inhibted memory is not possible, the device-driver must maintain cache-coherency with the BDs in memory by performing data cache-flushes and cache-line invalidates as appropriate while maintaining awareness that if the MAL is operatinal on the same channel that is being modified by software, incoherency can occur. This is possible because multiple BDs are present in a single cache-line. Therefore, cache-line invalidate and flush operations involve other BDs, and so a cache-line flush to force a BD write-back to memory, could corrupt other BD which may have been changed by MAL. Only by disabling the MAL so it can not be processing requests can cache-coherency with cached BDs be ensured.

Data buffers, in contrast, should be placed in cachable memory whenever possible as this greatly improves the speed of packet-processing by network-protocol or application software. The device-driver can easily ensure data-buffer cache-coherency of data-buffers by ensuring that:

- All buffers are aligned on a data cache-line size boundary

- All buffers are a multiple of a data cache line in size

Note: The data cache line-size PPC460EX/EXr/GT is 32 bytes. Therefore, the required buffer-alignment for cacheable buffers using is also 32 bytes. This also means that 4 BDs, which are 8 bytes each, fit into a cache-line.

27.3.4 Buffers and Cache-Coherency

When using cacheable memory for EMAC data buffers, the following practices must be followed:

Before setting the Empty bit in a RX BD, software must invalidate the all the cache-lines occupied by the buffer in the data-cache, as determined by the length specified in MAL register which configures the RX buffer size. This ensures that they are re-read in to the data-cache upon the next access by the CPU, and hence correctly contain the frames' data that MAL placed into them during reception.

Before transmitting a frame from a TX buffer, software must ensure that the desired data is in the TX buffer by flushing the involved cache-lines before setting the Ready bit in the TX BD.

27.4 Transmit Software Interface

Once a channel is enabled in MAL (this is done by setting the appropriate bit in the Channel Active Register), a channel may request service from MAL. When the first transmit request after a MAL reset comes in from a EMAC TX channel, MAL finds the starting address of the BD table for the channel by looking in the corresponding Transmit Channel Table Pointer Register and reads the first BD. If the first BD has a buffer-status of Ready, the MAL starts transferring the transmit-buffer contents to the EMAC.

When MAL begins processing a buffer, it writes the contents of the BD status/control field to the EMAC. This information configures the EMAC core for each Ethernet frame transmitted.

When all data from the current buffer has been transferred to the EMAC, MAL reads the next BD in the BD table and if Ready, again transfers the buffer's data to the EMAC.

When a BD indicates that it is the last buffer of the multi-buffer frame, MAL informs the channel that the last buffer's data transfer completes the transfer of an entire Ethernet frame. When EMAC finishes transferring the frame's data to the PHY, EMAC signals MAL to read the frame's transmit completion status. MAL then writes this information back into the status/control field of the last BD of the frame.

When the EMAC channel next requests that MAL process the next BD, the same frame handling process will be initiated. The first BD of the next frame immediately follows the descriptor marked "last" in the previous frame in the BD table.

27.4.1 Wrapping the BD Table (BDT) for Transmit

When MAL reads a BD (while handling a frame for a EMAC channel), it may encounter a Wrap indication within a BD control field. After MAL finishes processing the BD with he Wrap-bit set, it will "wrap around" to the first BD of in the BDT when processing the next BD. This wrap-around also happens when MAL has processed the 256th BD, because 256 is the maximum number of descriptors allowed in a BTD. The wrapping of the BDT, like all other BDT handling processes, is not signaled to the EMAC because it is not useful information for EMAC operation.

27.4.2 Continuous-Mode for Transmit

When the Continuous Mode (CM) bit is cleared in a buffer's BD status/control field, after transferring the data in the buffer, MAL clears the Ready bit in the BD control/status field. Clearing the Ready bit means MAL will not process the same BD again until software has refilled the buffer with valid data and reset the Ready bit. This is true even if only a single buffer is used for the channel.

When the Continuous Mode (CM) bit is set in a BD status/control field, MAL will not clear the Ready bit after transferring the buffer's data to the EMAC. This causes repeated transmission of the current data buffer without software intervention. This mode is generally used by protocols in which frequent re-transmission is an integral part of the protocol itself. In such cases, re-transmission can be performed without software intervention until software detects that the transmission has completed successfully and then stops transmission by disabling the MAL or EMAC.

27.4.3 Back-Up-a-Frame for Transmit

MAL is capable of re-transferring all the buffers of the current frame ("back-up-a-frame") upon request from an EMAC. In order to support re-transfer requests, the device-driver software must ensure that all the buffers of the re-transferred frame are still available with the correct content, that is that they were not released for use by any subsequent frames.

In normal buffer processing, MAL resets the Ready bit of a BD after finishing transfer of the buffers data to the TX FIFO. When MAL is requested to re-transfer the last frame, MAL doesn't reset the READY bit in the current buffer's descriptor, nor activate the end-of-buffer interrupt, nor write the EMAC's transfer completion-status word back to the buffer's descriptor. MAL also doesn't consider completion of transferring of this buffer as an end-of-buffer event.

On the next service request from the same channel, MAL will start transferring the current frame again, starting from the first buffer if it is a frame composed of more than one buffer.

Note: The currently processed BD can be either the last BD of the multi-buffer frame or, in case of early frame-termination, the BD, other than the last, that was being processed when the transmit channel initiated the early frame-termination.

During retransmission of a backed-up frame, MAL may use BDs on which the Ready bit was already cleared. MAL will retransmit the backed-up frame regardless of the Ready bit value. Therefore, the device-driver must not reuse BDs before the Ready bit of the 'last' BD is cleared as indicated by the Last bit of the MAL section of the status/control field of the BD.

Note: In the case of "descriptor-not-valid", that is a TX Descriptor Error for the first buffer of a frame, EMAC is not allowed to return a completion-status that contains a Back-Up-a-Frame request. That is it is not allowed to re-transfer a frame except after the first buffer has been transmitted.

27.4.4 Buffer Descriptor-Error Due to Not-Ready for Transmit

When MAL accesses a BD, the first check is whether or not the Ready bit is set. If the Ready bit is not set, one of three different events will occur depending on wether or not it is the first BD of a frame, or if the BD is one of a Backed-up frame:

Not a Backed-Up frame:

- If the BD is the first BD of the frame MAL informs the channel that data is currently unavailable. Further handling of this scenario is EMAC-specific. The channel might either instruct MAL to access the same BD periodically (by keeping its service request to MAL active) until it becomes ready, or 'give up' on the BD, completing the end-of-frame protocol with MAL. The channel might also indicate the BD status to the device driver via an

interrupt. However, in this case the EMAC should eventually complete the frame transfer protocol with MAL. Following a BD Not-Valid indication, MAL's BD pointer continues pointing to the same location in the BD table. The next time a BD read is initiated by the EMAC, the MAL searches for the buffer in the same location.

- If the BD is not the first BD of the frame, it causes a Descriptor-Error, i.e. a TXDE. MAL deactivates the channel and the processing of the current frame is aborted. Software may learn about this error from either one of two possible MAL interrupts (or from both).
  - The first possible interrupt request is non-maskable at the MAL and indicates the TX channel involved, because there is a separate interrupt bit for each TX channel. See *Descriptor Error Interrupt Registers (MAL0_TXDEIR, MAL0_RXDEIR)* on page 984).
  - The second one is a maskable interrupt which indicates a BD error event, regardless the channel number (one interrupt bit for all the channels, see *MAL Error Status Register (MAL0_ESR)* on page 981). For more information about error handling, see *MAL Error Handling* on page 965.

It is Backed-Up frame:

- When the current transmitted frame is a backed-up frame, all BDs except the last, are valid even though the READY bit is not set. In this case, excepting the Last BD (Last bit is set), MAL processes the frame's BDs regardless of the READY bit value. If the READY bit of the last BD in the backed-up frame is not set, MAL treats it as a Descriptor-Not-Valid error. MAL handles the BD error as described above for the case when the frame isn't a backed-up frame.

27.4.5 Advance to First Buffer-Descriptor of a Frame (Scroll) for Transmit

In case of early-frame-termination, MAL may be configured to advance (scroll) through BDs in the BD table to the first BD of the next frame.

When a multiple-buffer frame is terminated early by the EMAC, while the MAL is processing a buffer which is not the last buffer in the frame, the MAL can operate in one of the following ways:

The "Scroll through remaining BDs" bit in the MAL configuration register is set:

- In this case the MAL reads the completion status-word from the EMAC channel. Then the MAL resets the READY bit and writes the completion-status word to all the remaining BDs of the current frame as determined by the setting of the Last bit in the BD. On the next service of this channel, the MAL fetches the first BD of the next frame.

The "Scroll through remaining BDs" bit in the MAL configuration register is clear:

- In this case the MAL reads the Status word from the EMAC channel. Then the MAL terminates the current channel service by resetting the READY bit of the last processed BD (the one in which there was an early termination) and writes the status only to this BD. On the next service of this channel, MAL fetches the next BD in the current frame. In this case, the software is responsible to monitor the MAL location in the BD table.

In the case that the EMAC requests a re-transmit of the early terminated frame (when the "backup" bit in the EMAC status is set), MAL re-transmits the frame regardless of the setting of MAL Scroll Descriptor bit.

27.5 Receive Software Interface

MAL uses the RX channel BDs in a manner similar to that used for transmission. Once an RX EMAC channel requests that a new frame be processed, MAL starts processing the channel's next BD in the table. Once a channel is enabled in MAL, the channel may request MAL service. When it does, MAL accesses the first BD (in that channel's BD table) that is pointed to by the EMAC channel table pointer register. If that descriptor is ready (empty for RX), MAL starts processing the buffer.

When it begins processing each frame, MAL writes the contents of the status/control field into the EMAC. This information can be used by the EMAC for a per-frame configuration.

Once data is received from the memory, MAL moves the data from the RX channel FIFO into the data buffer pointed to by the first BD. The current BD may be closed for two reasons: there is no more room left in the buffer, or the EMAC channel indicated that the frame reception ended. If additional buffering space is needed for the current frame, MAL moves on to the next BD. As each BD is closed, MAL updates the length field with the actual amount of bytes written into the buffer. The maximal length of the buffers for each channel is defined by a configuration register (see *RX Channel Buffer Size Register (MAL0_RCBSx)* on page 988). The maximal receive buffer length is defined per channel.

Once the EMAC channel indicates that the frame reception has ended, it is expected to request that MAL update the received frame status in the BD status/control field. MAL updates the frame status and notifies the EMAC. At this point the frame is considered received and the EMAC may request that MAL begin the process of receiving a new frame. The first buffer of the next frame is the buffer in the BD table that followed the last descriptor of the previous frame.

27.5.1 Wrapping the BD Table for Receive

When MAL processes the BD, it may encounter a Wrap indication within a BD control field. This causes MAL to go back to the head of the channel's BD for the next BD. This also happens when MAL reaches the maximal number of descriptors.

27.5.2 Continuous Mode for Receive

After using a BD, MAL sets the BD control to the Not-Empty state. In this way, MAL will not use the same BD a second time until the software has processed the not-empty BD and set it to Empty again. MAL will not clear the Empty bit while the Continuous Mode (CM) bit is set in the status/control field. The Continuous Mode is generally used by protocols where frequent collisions are an integral part of the protocol itself (forcing the EMAC to abort a reception process and restart). In such cases, re-reception can be performed without software intervention.

27.5.3 Descriptor Not Valid for Receive

When MAL accesses a BD it may find that the Empty bit is not set. In the case of an RX channel descriptor, this situation is considered as a descriptor error. MAL deactivates the channel and from its point of view, the processing of the current frame has ended. Software may learn about this situation from one of two MAL interrupts (or from both):

- An RXDE interrupt with the MAL0_RXDEIR indicating which channel caused the interrupt
- An SERR interrupt (system error) with one interrupt bit for all channels in the MAL0_ESR

For more about error handling, see *MAL Error Handling* on page 965.

27.5.4 Buffer Length for Receive

The maximum length of an RX buffer is the same for all RX buffers in each channel. The value is programmable through a set of MAL registers (see *MAL Registers* on page 973). The actual data length used by data placed into the RX buffer is written by MAL into the buffer-descriptor upon completion of packet transfer. If the buffer is completely filled up, the value written will match the maximum value programmed into the channel's RX-Channel-Descriptor data-length register. If the buffer is only partially filled, which occurs when the RX frame ends before using the buffer's complete space, the actual amount of space filled is written into the buffer-descriptor length field.

27.6 Buffer Descriptor Status/Control Fields

The following sections detail the status/control field bits. The information fields within the status/control field can be divided as follows:

- Information from a software device driver directed to MAL and EMAC
- Information from MAL and EMAC directed to software
- Status/control field handling
- Status/control field format
- TX status/control field format
- RX status/control field format

27.6.1 Information from a Software Device Driver Directed To MAL and EMAC

- MAL-related BD processing information:
  - Buffer Ready/Not Ready (determines the buffer's validity).
  - Wrap to top of table or continue to next descriptor.
  - In a transmit BD – Is the current buffer the last one in the frame?
  - Continuous or normal mode – Should MAL change the Ready/Not Ready value?
  - Should the channel generate an interrupt following the end of frame processing?

- EMAC channel configuration information:
  - Protocol specific configuration.

27.6.2 Information from MAL and EMAC Directed to Software

- MAL generated status information:
  - Buffer Ready/Not Ready (passes the buffer handling to software).
  - In receive BD - Is the current buffer the first one in the frame?
  - In receive BD - Is the current buffer the last one in the frame?
- EMAC channel generated status information:
  - Protocol specific error and status information (transmit and receive).

27.6.3 Status/Control Field Handling

When MAL accesses a new BD, the status/control word is written to the EMAC channel. This allows the channel to configure itself for the current frame.

For all "intermediate" BDs (all descriptors that do not contain the frame's ending), the status/control field is written by MAL (rather than the EMAC). In this case, the status/control field indicates that the current buffer is not the last one in the current frame.

As MAL finishes processing the last BD in a given frame, it reads the channel's status (via an OPB transaction) and writes it into the BD's status/control field.

In effect, since all of the various control and status fields do not overlap, the status/control halfword is read/written as a whole. Each agent (MAL, EMAC channel, and software) reads the entire status/control halfword, relates to specific fields of interest, and updates another subset of fields within the same halfword. While an agent modifies its related fields, all other fields remain unchanged.

27.6.4 Status/Control Field Format

The status/control halfword is divided into EMAC channel data and MAL related data. As explained above, the MAL related fields are either aimed at controlling MAL or written by MAL for use by the software. The MAL fields are of no interest to the EMAC (except the Ready and Empty bits).

The same applies to the EMAC channel fields. The EMAC related fields are either aimed at controlling the EMAC or written by EMAC for use by the software. These fields are of no interest to MAL.

MAL will not manipulate the EMAC related fields, and EMAC is not allowed to manipulate the MAL related fields.

27.6.5 TX Buffer Descriptor MAL Control

The bit numbering in Table 27-2 relates to a Buffer Descriptor's (BD's) lowest addressed 16-bit field, the Control and Completion Status field.

Table 27-2. TX Buffer Descriptor Status Control Fields

0	R	Ready 0 Not ready 1 Ready	The device driver sets Ready after preparing the buffer for transmission. As soon as Ready is set, MAL may process the buffer at anytime, therefore software should must not do any further writing to any field of the BD or the associated buffer. MAL clears this bit when finish processing the BD. MAL doesn't clear the Ready bit in the case of backing-up to retransmit a frame in the case of collision or in continuous-mode (see *Back-Up-a-Frame for Transmit* on page 957 and *Continuous-Mode for Transmit* on page 957). Note: BDs can not be cached as there is no mechanism to maintain cache cohesion when MAL writes to it to update the completion status.
1	W	Wrap 0 This is not the last data BD in the BD table. 1 This is the last data BD in the BD table.	If W=1, after this buffer has been processed, the MAL next processes the first buffer-descriptor in the table. This bit is controlled by software only.and does not affect the EMAC operation.
2	CM	Continuous Mode 0 Normal Operation 1 Continuous Operation.	CM=0: After buffer is processed, the Ready bit is reset. CM=1: After this buffer is processed, the Ready bit is not reset to 0 by the MAL. This means that the buffer remains ready for transmission so that when the MAL next accesses this BD it is transmitted again. The Ready-bit is cleared if an error occurs during transmission regardless of the setting of CM. This bit is controlled by software only and does not effect EMAC operation.
3	L	Last 0 Not the last buffer in the frame being processed. 1 Last buffer of the frame.	Controls the MAL's signaling of end-of-frame to EMAC. When L=1 the MAL signals end-of-frame to the EMAC after transferring the last byte of this buffer to the EMAC TX FIFO. This bit is controlled by software only.
4		Reserved	This field must be set to zero by software.
5	I	Interrupt 0 No IRQ is generated. 1 Last buffer of the frame.	MAL asserts the End Of Buffer interrupt after it updates the BD's completion-status field. This bit is controlled by software only. It controls MAL operation, and does not affect the EMAC operation.
6:15		EMAC Control and Completion Status	This field is used by the EMAC. It contains control fields configured by software in order to control the per-frame operation of the EMAC, and to provide EMAC's frame transmission completion-status to the device driver.

27.6.6 RX Buffer Descriptor MAL Control Options

The bit numbering in *Table 27-3* relates to the buffer descriptors (BD's) fullword which contains both the status/control and the length fields.

Table 27-3. RX Buffer Descriptor MAL Control Fields

0	E	Empty 0 Not Empty 1 Empty	Software sets this bit when the buffer is available for reception of frame data. MAL clears this bit after the buffer has been used to receive data, or after an error is encountered when some frame data has been placed into the buffer. Note: While E=0 software is free to write to any fields of this BD since the MAL does not process it. While E=1, this buffer-descriptor and its associated data-buffer may be used by MAL at anytime. Therefore once the E-bit is set, software must not write to any fields of this BD. Note: See bit 2 - Continuous Mode, below. Note: BDs can not be cached as there is no mechanism to maintain cache cohesion when MAL writes to it to update the completion status.
1	W	Wrap 0 Do not wrap 1 Wrap	W=0: Process the next BD in the BD table. W=1 After this buffer has been processed, the MAL uses the first BD of the BD table. This bit is controlled by software only. It controls MAL operation, and does not affect the EMAC operation.
2	CM	Continuous Mode 0 Normal Operation 1 Continuous Operation	CM=0: After buffer is used the Empty bit is reset. CM=1: After this BD is processed, the Empty-bit is not cleared by MAL. This means that the data buffer remains ready to receive data and is reused by the MAL when the MAL next accesses this BD. The Empty-bit is cleared if an error occurs during reception regardless of the setting of CM. This bit is controlled by software only. It controls MAL operation, and does not affect the EMAC operation.
3	L	Last 0 Not the last buffer of the multi-buffer frame. 1 Last buffer of the multi-buffer frame.	Updated by the MAL following the use of the buffer. Controlled by EMAC's signaling of End-OF-Frame to the MAL. When the End-Of-Frame is detected by the EMAC, it signals End-Of-Frame to the MAL while transferring the last data of this frame to the MAL.
4	F	First 0 Not the first buffer of the multi-buffer frame. 1 First buffer of the multi-buffer frame.	Updated by the MAL following the use of the buffer. Controlled by EMAC's signaling of Start-Of-Frame to the MAL. When the End-Of-Frame is detected by the EMAC it signals End-Of-Frame to the MAL while transferring the last data of this frame to the MAL.
5	I	Interrupt 0 There is no action taken by MAL once it reaches the end of the current buffer. 1 After finishing processing the current buffer, if this bit is 1, the end of buffer field in the End of Buffer Interrupt Status Register is set and the end of buffer interrupt is asserted.	MAL asserts the End Of Buffer interrupt after it updates the BD's completion-status field. This bit is controlled by software only. It controls MAL operation, and does not affect the EMAC operation.
6:15		EMAC Control and Completion status	This field is only used by the EMAC. It contains control fields configured by software in order to control the per-frame operation of the EMAC, and to provide EMAC's frame transmission completion-status to the device driver.

27.7 MAL Programming Notes

The following sections contain information about programming the MAL.

27.7.1 MAL Initialization

MAL initialization includes two parts: configuration and channel activation.

Configuration involves two steps:

- MAL configuration - This step is done only after a power on reset or after a MAL soft reset. The following registers are involved:
  - Configuration Register (MAL0_CFG). This register defines MAL operation on the PLB and OPB.
  - Interrupt Enable Register (MAL0_IER). This register is used to enable interrupts for various MAL error conditions.
- Channel specific configuration - This information may be changed only when the associated channel is not active. MAL0_TXBADDR and MAL0_RXBADDR must be written before writing to the MAL0_TXCTPxR and MAL0_RXCTPxR registers. (The bit for the channel in the TX or RX Channel Active Set Register, is cleared.) The following registers are involved:
  - MAL0_RCBSx – RX Buffer Size (eight registers for each physical RX channel). This register defines the length of the RX buffers in memory.
  - MAL0_TXCTPxR or MAL0_RXCTPxR – Channel Table Pointer Register (one register for each TX channel and eight registers for each physical RX channel). This register is programmed with the memory address of the first BD table entry for the channel.

Setting the channel specific configuration can be done as part of MAL initialization or as part of the EMAC initialization process. In order to activate a channel, the following actions should be taken:

- The channel has to be configured in MAL
- The related bit in Channel Active Set Register (MAL0_TXCASR or MAL0_RXCASR) has to be set
- The channel operation must be enabled (EMAC configuration)

27.7.2 Interrupts

The MAL in the PPC460EX/EXr/GT has five interrupt lines which are connected to the Universal Interrupt Controller (*Interrupt Controller Operations* on page 255) Two interrupt lines, one for TX and one for RX, are used for interrupt events during frame transfer. An additional two interrupt lines, one for TX and one for RX, are used to report descriptor errors on a per-channel basis. The fifth interrupt is used to report MAL errors.

- TXEOB interrupt line is used to report end of buffer or end of frame for a specific TX channel. A bit for the related channel is set in the MAL0_TXEOBISR. See *End of Buffer Interrupt Status Registers (MAL0_TXEOBISR, MAL0_RXEOBISR)* on page 980.
- RXEOB interrupt line is used to report end of buffer or end of frame for a specific RX channel. A bit for the related channel is set in the MAL0_RXEOBISR. See *End of Buffer Interrupt Status Registers (MAL0_TXEOBISR, MAL0_RXEOBISR)* on page 980.
- TXDE interrupt line is used to indicate a descriptor error event in a specific TX channel descriptor table. A bit for the related channel is asserted in the MAL0_TXDEIR. See *Descriptor Error Interrupt Registers (MAL0_TXDEIR, MAL0_RXDEIR)* on page 984.
- RXDE interrupt line is used to indicate a descriptor error event in a specific RX channel descriptor table. A bit for the related channel is asserted in the MAL0_RXDEIR. See *Descriptor Error Interrupt Registers (MAL0_TXDEIR, MAL0_RXDEIR)* on page 984.

- SERR interrupt is used to report a system error indicated by MAL. For more information on handling the SERR interrupts, see *MAL Error Handling* on page 965.

27.7.3 MAL Error Handling

MAL communicates with the EMACs using the EOPB; and with buffer memory, typically DRAM, via the PLB. As long as this bus communication is error-free and no buffer-descriptor errors are detected, MAL continues to service transfer-requests for the channels enabled by the MAL0_TXCASR and MAL0_RSCASR Channel-Active Set registers. Errors may occur because of MALs' communication with the EMAC via the EOPB; on the PLB usually due to invalid memory access attempts by the MAL, from buffer-descriptor errors, or from accessing the buffers themselves. MAL handles errors on a per-channel basis when possible, but some errors can not be attributed to a specific channel.

When a PLB error occurs, MAL is notified by the PLB-Error signal input - which is asserted by a PLB slave, typically the DRAM controller. When a EOPB error occurs, MAL is notified by the EOPB-Error signal from an EMAC. The MAL can also be notified of errors by either buses arbiter, typically when a transaction-initiation timeout occurs because no slave claimed the transaction due to using an invalid address.

EOPB errors are normally attributable to a specific EMAC and its associated TX or RX service-channel; therefore only the involved channels' operation is stopped. When a buffer-descriptor error or buffer-access error occurs, the channel involved is known because each channel has its own buffer-descriptor table; so only the specific channel is stopped.

When an error is signaled to the MAL while MAL is performing a data transfer, if the identity of the channel can be known, then MAL immediately aborts the transfer for the specific channel and does not respond to further service requests from that channel. It does this by disabling and resetting the channels' operation by clearing the appropriate bit in the MAL0_TXCASR for transmission, or MAL0_RSCASR for reception. This, in effect, records the channel(s) experiencing errors and that have been made inactive. It also keeps a record of the cause of the error. MAL also asserts a (maskable) interrupt request signal to the UIC core.

In order to re-enable service requests for a channel, software must set the channels' bit in the Channel-Active register. When the channel is reactivated and a transfer is requested, the MAL resets its buffer-descriptor counter and therefore use the first buffer-descriptor in the buffer-descriptor table.

In the case of a PLB-error MAL cannot identify which channel is involved, therefore no channels' operation is disabled and MAL remains ready to attempt to service further requests from all EMACs. PLB-error handling and channel de-activation (if required) are the responsibility of the software.

Note 1: Device-driver software must configure the MAL before processing service requests from EMAC's. When handling errors, if the MAL requires reconfiguration, the device-driver should ensure that no MAL channels are enabled during the reconfiguration or further errors may occur.

Note 2: When one virtual channel encounters a problem, the physical RX channel halts. Once the physical channel halts all eight virtual channels associated with this physical RX channel halt as well.

27.7.3.1 MAL Error Causes

The cause of a MAL error is stored in the Error Status Register, MAL0_ESR.

- **PLB: Buffer-Descriptor Error**

  If a buffer is not Ready or Empty when its descriptor is accessed by MAL, it causes a buffer-descriptor error. MAL also deactivates the channel when it detects that the associated buffer is not ready for use in a transfer of data to/from the EMAC.

For receive (RX) channels, a buffer-descriptor error occurs when:

- MAL accesses a buffer-descriptor in which the Empty bit is zero, that is the buffer is already filled with previously received data.

For transmit (TX) channels, a buffer-descriptor error occurs when:

- MAL processes a buffer-descriptor in which the Ready bit is zero, that is the buffer does not have data ready for transmission *except:*
  - if it is the first buffer-descriptor of a TX frame because it is normal that after processing TX buffers that are filled an empty buffer is accessed.
  - or the access is to a buffer-descriptor that is not the last buffer-descriptor in a backed-up-for-retransmission multi-buffer frame.

As a result of a buffer-descriptor error, the following actions are taken by MAL without software assistance:

- The Active bit of the related channel in the Channel Active register is zeroed, thereby halting channel activity and disabling servicing of subsequent data transfer requests.
- The associated bit in the TX Descriptor-Error Interrupt Register (MAL0_TXDEIR) or RX Descriptor-Error Interrupt Register (MAL0_RXDEIR) is set, causing a non-maskable (at the ESR register level) TX Descriptor Error (TXDE) or RX Descriptor Error (RXDE) interrupt request. Then, depending on the mask defined in MAL0_IER (Interrupt Enable Register), it sends an interrupt to the Universal Interrupt Controller. The UIC input to which this IRQ is connected can mask the IRQ to the processor core.

- **OPB: Non-Fullword Error**

  This error indicates that the 'non-fullword acknowledge' signal was asserted by a EMAC.

  The Active bit of the related channel in the Channel Active register is zeroed, thereby halting channel activity and disabling servicing of subsequent data transfer requests.

- **OPB: Time-Out Error**

  This error indicates that the OPB Time-out error was reported by the OPB arbiter because no EMAC, which are EOPB slaves, claimed the transfer transaction. This only occurs in case of MAL or EMAC hardware malfunction.

  The Active bit of the related channel in the Channel Active register is zeroed, thereby halting channel activity and disabling servicing of subsequent data transfer requests.

- **OPB Error**

  This error indicates that the OPB-Error signal was asserted by the EMAC because although it claimed the transaction, the EMAC could not complete it.

  The Active bit of the related channel in the Channel Active register is zeroed, thereby halting channel activity and disabling servicing of subsequent data transfer requests.

- **PLB Error**

  This error indicates that the PLB Error signal was asserted by the PLB slave, typically the DRAM controller since the buffers and their descriptors are normally created in the main system memory.

  In this case, MAL cannot determine which channel caused the error so operation is not halted for any of the channels, that is no Active bits in the Channel Active register are zeroed.

27.7.3.2 Error Handling Registers

MAL error handling logic affects two registers; Error Status (ESR) and Channel Active (TXCASR and RXCASR).

- **Error Status Register (ESR)**

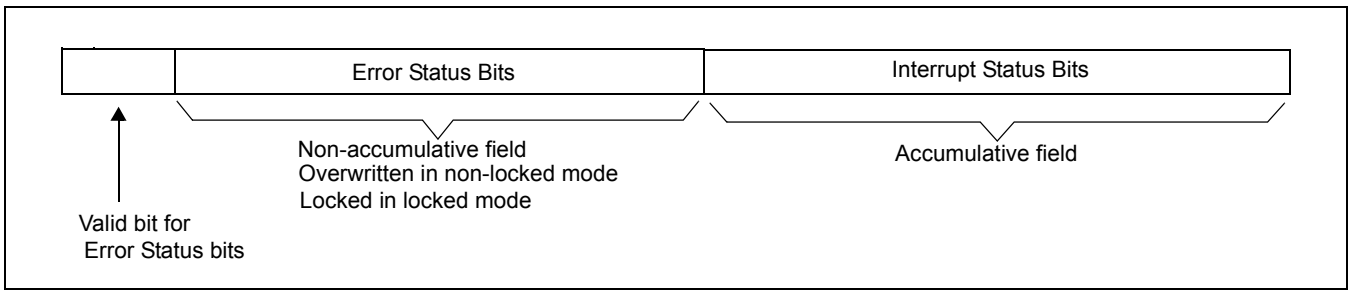
  This register holds information about the cause of the error that occurred and the interrupt status. The register includes the following fields:

  Error status – This field holds the error cause information. The information includes the number of the channel on which the error occurred (if known) and the cause of the error. The error can be either the last detected error, or the first error detected if "Locked error mode" is active. See *Operational Error Modes* on page 967 for description of the Locked error mode.

  The error status field includes an "Error Valid" bit which indicates whether there is valid error information in the error status field or not. The error status field is not valid when the "Error Valid" bit is cleared (by writing 1 to this bit).

  Interrupt status – Every error detected causes MAL to set a bit in the interrupt status field. Software can clear an interrupt request bit by writing 1 to the bit to be cleared in the interrupt status field. The bits in this field are not reset except by software writing a 1 to the bit, which allows more than one interrupt to be indicated simultaneously.

Figure 27-5. Error Status Register Field



27.7.3.3 Operational Error Modes

MAL can operate in two different error handling modes:

- Locked Error Mode: Information about the error is written to the Error Status Register, and the Valid bit in that register is set. Information in the Error Status field of the register stays locked until software unlocks it by resetting the error Valid bit. The Interrupt Status bits of the Error Status Register are not locked in this mode, so software can find out if more errors occur. However, the Error Status field applies only to the first error that is locked.
- Non-Locked Error Mode: Information about the error is written in the Error Status Register, and the error Valid bit is set. Each new error is overwritten, so the information in the Error Status Field is valid only for the last error that occurred.

In both modes, each error written in the error description field sets the error Valid bit, and it is the responsibility of software to reset this bit.

The error handling mode is programmed in the MAL Configuration Register (see *MAL Configuration Register (MAL0_CFG)* on page 977).

27.7.3.4 Resolution of an Error Situation

After receiving an interrupt from MAL, software must analyze the error information read from the Error Status Register to determine what action to take. Software can restart channel activity by setting the associated bit in the Channel Active Register.

When a channel is activated, the MAL starts processing descriptors from the first descriptor in the channel descriptor table. Therefore, software must also update the value of the other channel related registers (see *Channel Table Pointer Registers (MAL0_TXCTPxR, MAL0_RXCTPxR)* on page 986) in order to continue from the same buffer in memory, when the error is not buffer related.

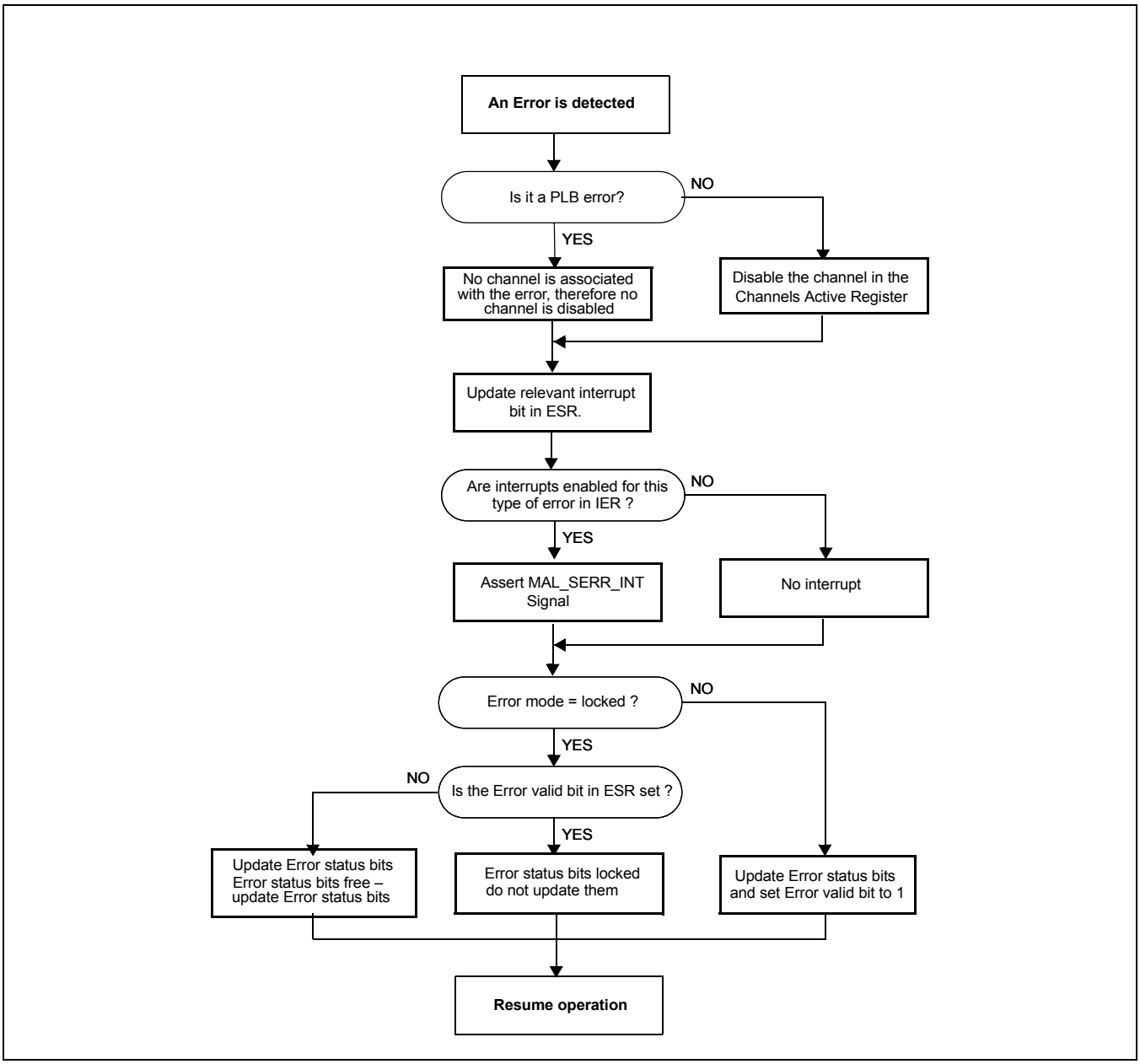
In the case of PLB errors, MAL does not know which channel caused the error. It is the responsibility of the software to analyze the MAL error registers and the PLB slave error registers to determine which channel caused the error. Software should deactivate the channel, resolve the problem, and then reactivate the channel.

See *MAL Error Processing* on page 969 for a flow chart illustrating the steps MAL performs when handling an error.

27.7.3.5 Interrupts To Software

Table 27-6 describes MAL actions once an error is detected. Note that the actual decisions MAL makes may be in a different order than represented by this figure. The device driver should realize that all of the MAL error-handling actions are completed before the error is processed by software at the "Resume Operation" block.

Figure 27-6. MAL Error Processing



27.8 Transmit and Receive Operations

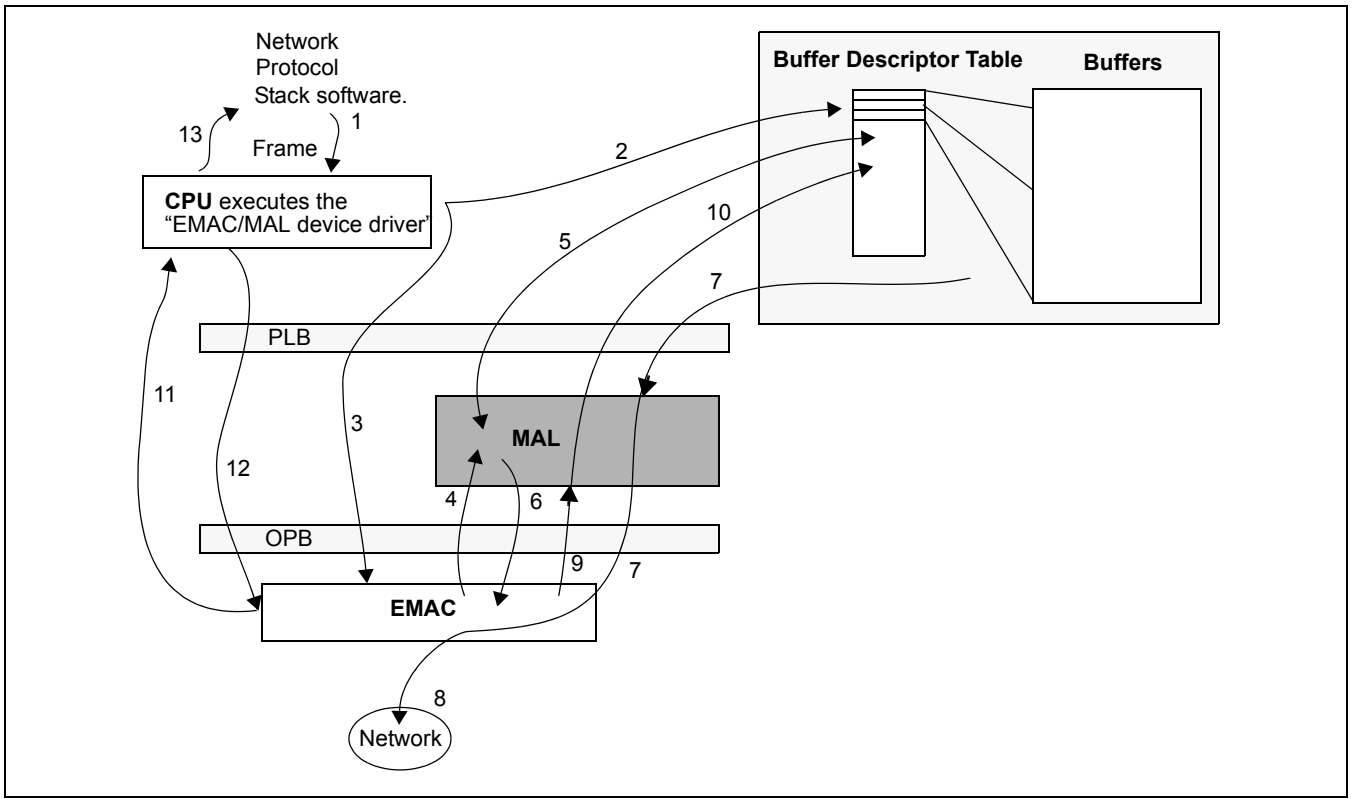
The device-driver is responsible for configuring MAL before a EMAC can begin requesting MAL services to transfer data to/from the EMAC cores. The device-driver must ensure that channels are not enabled during reconfiguration; otherwise, fatal errors may occur.

For more detail about MAL software requirements, see *MAL Programming Notes* on page 964.

27.8.1 Transmit Operations

Figure 27-7 describes the operation of software and hardware in order to transmit a packet of data.

Figure 27-7. Transmit Operations



The steps illustrated in the figure above are performed in numerical order:

1. The networking-services protocol-stack software (application-level software) makes a packet/segment transmit request to the MAL/EMAC device-driver.
2. The MAL/EMAC device-driver software parses the protocol-stack packet buffer into as many MAL buffers as required and configures each buffers associated buffer-descriptor.
3. The device-driver signals the EMAC to process a new pending transmit buffer by setting the Get-Next-Packet bit in the EMACs' control register.
4. The EMAC uses its channel with the MAL to request data transfer-service from the MAL to transfer the frames' buffer data to the EMAC.
5. MAL checks that the buffer-descriptor is available for transmission by seeing that the Ready bit is set, and if it is, MAL fetches the EMAC configuration-word from the buffer-descriptor control/status field.
6. MAL writes the configuration-word into the EMACs control register.

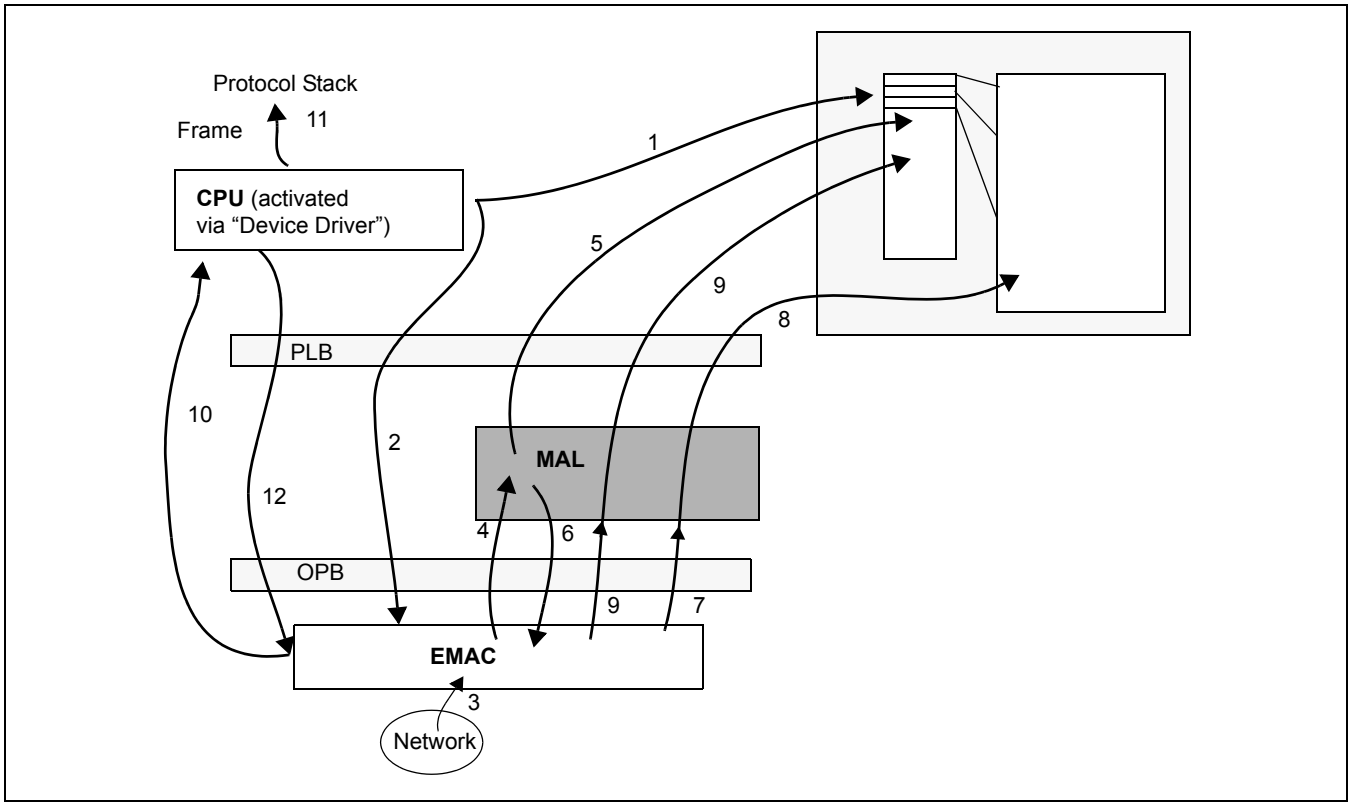
7. MAL transfers the buffers' data from memory to the EMACs TX FIFO. The EMAC controls the pace of the data transfer to the TX FIFO as configured by the EMACS service-request thresholds.
8. Once the EMACs TRTR threshold is reached, data from the TX FIFO is transmitted to the PHY to be placed onto the media. Steps 7 and 8 occur simultaneously.
9. Upon completion of the buffers' transmission to the PHY the EMAC requests that MAL read the buffers data-transfer completion-status word.
10. The completion-status word read by MAL from the EMAC in step 10 is written to the buffers' descriptor.
11. Optionally, an IRQ is generated (if interrupt conditions are met) by the EMAC and/or by the MALs' end-of-buffer interrupt event.
12. The device-driver clears the IRQ status bits to negate the IRQ in the EMAC and in MAL as necessary.
13. The device-driver informs the networking-services protocol-stack software that transmission is complete. Typically, the stack assumes the frame is transmitted after step 1. But in this step the protocol-stack software may do other processing such as releasing the buffer for reuse by returning it to the queue of empty protocol-stack buffers.

Note: The description in *Figure 27-7* is a general description for MAL and EMAC usage by software. The device-driver should follow all recommendations given in the section *General PPC460EX/EXr/GT Structure* on page 949.

27.8.2 Receive Operations

Figure 27-8 describes the software and hardware operations when receiving a typical frame.

Figure 27-8. Receive Operations



The steps above are performed in the order described below:

1. The MAL/EMAC device-driver software creates a pool of receive buffers and configures the RX channel buffer-descriptor table to use them.
2. The device-driver configures the EMAC to generate requests for transfer-service by the MAL.
3. The EMACs RXMAC logic receives data from the PHY and places it into the RX FIFO. Steps 2 and 3 can be reversed in order.
4. Once the request-for-MAL-service threshold is reached, the EMAC requests transfer-service from the MAL.The EMAC uses its' MAL RX service-request channel on the EOPB to request that MAL transfer the frame data to buffer-memory via the PLB.
5. Upon the EMAC winning the service-request from the MALs RX service-request arbiter, MAL fetches the EMAC RX configuration word from the current buffer-descriptor.
6. MAL writes the EMAC configuration-word to the EMAC.
7. MAL then performs the data transfer by reading the frames' data from the RX FIFO using the EOPB.
8. MAL writes the frames' into the buffers memory using the PLB.
9. The EMAC continues to receive data from the PHY and places it into the RX FIFO. Once the request-for-MAL-service threshold is again reached, the EMAC requests transfer-service from the MAL. MAL then performs steps 7 and 8. When the a entire frames worth of data has been transferred into one or more receive buffers,

MAL reads transfer-completion status information from the EMAC RXMAC logic and writes it to the last buffer's buffer-descriptor status/control word field.

10. The device-driver is interrupted (if interrupt conditions are met) by the EMAC and/or by the MAL end-of-buffer interrupt event.
11. The received frame is passed to the networking-protocol stack software.
12. Once the packet is processed or copied to another protocol-stack buffer, software sets the Empty bit(s) in the buffer-descriptor(s), thus allowing them to be used again by the MAL.

Note: The description in *Figure 27-8* is a general description of MAL and EMAC operation and requirements of software. However, the device-driver should follow all recommendations given in *MAL Programming Notes* on page 964.

27.9 MAL Registers

The MAL registers are Device Control Registers (DCRs).

- Unless otherwise specified, all register fields are initialized at chip reset to 0.
- Reserved fields are read as undefined and must be written as 0s.

Table 27-4. MAL DCR Summary

Mnemonic	Register	DCR Number	Access	Page
MAL0_CFG	Configuration	0x180	R/W	977
MAL0_ESR	Error Status Register	0x181	R/Clear	981
MAL0_IER	Interrupt Enable Register	0x182	R/W	983
MAL0_TXCASR	Transmit Channel Active Set Register	0x184	R/W	978
MAL0_TXCARR	Transmit Channel Active Reset Register	0x185	R/W	978
MAL0_TXEOBISR	Transmit End of Buffer Interrupt Status Register	0x186	R/Clear	980
MAL0_TXDEIR	Transmit Descriptor Error Interrupt Register	0x187	R/Clear	984
MAL0_TXBADDR	Transmit Descriptor Base Address Register	0x189	R/W	986
MAL0_RXCASR	Receive Channel Active Set Register	0x190	R/W	978
MAL0_RXCARR	Receive Channel Active Reset Register	0x191	R/W	978
MAL0_RXEOBISR	Receive End of Buffer Interrupt Status Register	0x192	R/Clear	980
MAL0_RXDEIR	Receive Descriptor Error Interrupt Register	0x193	R/Clear	984
MAL0_RXBADDR	Receive Descriptor Base Address Register	0x195	R/W	986
MAL0_TXCTP0R	Channel Transmit 0 Channel Table Pointer Register	0x1A0	R/W	986
MAL0_TXCTP1R	Channel Transmit 1 Channel Table Pointer Register	0x1A1	R/W	986

Table 27-4. MAL DCR Summary (Continued)

Mnemonic	Register	DCR Number	Access	Page
MAL0_RXCTP0:15R	Channel Receive 0:15 Table Pointer Registers	0x1C0–0x1CF	R/W	986
MAL0_RCBS0:15	Channel Receive 0:15 Buffer Size	0x1E0–0x1EF	R/W	988

27.9.1 MAL Register to Channel Mapping

Table 27-5 indicates how TX and RX channels map to MAL register bit fields. When VLAN is not used or when QoS User Priority is set to 0, RX virtual channel 0 is the default for EMAC0, RX virtual channel 8 is the default for EMAC1, RX virtual channel 16 is the default for EMAC3 and RX virtual channel 24 is the default for EMAC3. See *Section 28.9.2 VLAN Tagged Packet Reception* for addition details about QoS.

Table 27-5. MAL Register to Channel Mapping

Register	Field	Bit(s)	Channel	Comment
MAL0_TXCASR	TCAS	0	TX Channel 0	EMAC0 TX (physical TX channel 0)
		1	TX Channel 1	EMAC1 TX (physical TX channel 1)
		2	TX Channel 2	EMAC2 TX (physical TX channel 2)
		3	TX Channel 3	EMAC3 TX (physical TX channel 3)
MAL0_TXCARR	TCAR	0	TX Channel 0	EMAC0 TX (physical TX channel 0)
		1	TX Channel 1	EMAC1 TX (physical TX channel 1)
		2	TX Channel 2	EMAC2 TX (physical TX channel 2)
		3	TX Channel 3	EMAC3 TX (physical TX channel 3)
MAL0_TXEOBISR	TCEI	0	TX Channel 0	EMAC0 TX (physical TX channel 0)
		1	TX Channel 1	EMAC1 TX (physical TX channel 1)
		2	TX Channel 2	EMAC2 TX (physical TX channel 2)
		3	TX Channel 3	EMAC3 TX (physical TX channel 3)
MAL0_TXDEIR	TXDE	0	TX Channel 0	EMAC0 TX (physical TX channel 0)
		1	TX Channel 1	EMAC1 TX (physical TX channel 1)
		2	TX Channel 2	EMAC2 TX (physical TX channel 2)
		3	TX Channel 3	EMAC3 TX (physical TX channel 3)
MAL0_TXCTP0R		0:31	TX Channel 0	EMAC0 TX (physical TX channel 0)
MAL0_TXCTP1R		0:31	TX Channel 1	EMAC1 TX (physical TX channel 1)
MAL0_TXCTP2R		0:31	TX Channel 2	EMAC2 TX (physical TX channel 2)
MAL0_TXCTP3R		0:31	TX Channel 3	EMAC3 TX (physical TX channel 3)
MAL0_RXCASR	RCAS	0	RX Channel 0	EMAC0 RX (physical RX channel 0)
		1	RX Channel 1	EMAC1 RX (physical RX channel 1)
		2	RX Channel 2	EMAC2 RX (physical RX channel 2)
		3	RX Channel 3	EMAC3 RX (physical RX channel 3)

Table 27-5. MAL Register to Channel Mapping (Continued)

Register	Field	Bit(s)	Channel	Comment
MAL0_RXCARR	RCAR	0	RX Channel 0	EMAC0 RX (physical RX channel 0)
		1	RX Channel 1	EMAC1 RX (physical RX channel 1)
		2	RX Channel 2	EMAC2 RX (physical RX channel 2)
		3	RX Channel 3	EMAC3 RX (physical RX channel 3)
MAL0_RXEOBISR	RCEI	0:7	RX Channel 0-7	Virtual Channel 0–7 EMAC0 RX (physical RX channel 0)
		8:15	RX Channel 8-15	Virtual Channel 8–15 EMAC1 RX (physical RX channel 1)
		16:23	RX Channel 16-23	Virtual Channel 16–23 EMAC2 RX (physical RX channel 2)
		24:31	RX Channel 24-31	Virtual Channel 24–31 EMAC3 RX (physical RX channel 3)
MAL0_RXDEIR	RXDE	0:7	RX Channel 0-7	Virtual Channel 0–7 EMAC0 RX (physical RX channel 0)
		8:15	RX Channel 8-15	Virtual Channel 8–15 EMAC1 RX (physical RX channel 1)
		16:23	RX Channel 16-23	Virtual Channel 16–23 EMAC2 RX (physical RX channel 2)
		24:31	RX Channel 24-31	Virtual Channel 24–31 EMAC3 RX (physical RX channel 3)
MAL0_RXCTP0R		0:31	RX Channel 0	Virtual Channel 0–7 EMAC0 RX (physical RX channel 0)
MAL0_RXCTP1R		0:31	RX Channel 1	
MAL0_RXCTP2R		0:31	RX Channel 2	
MAL0_RXCTP3R		0:31	RX Channel 3	
MAL0_RXCTP4R		0:31	RX Channel 4	
MAL0_RXCTP5R		0:31	RX Channel 5	
MAL0_RXCTP6R		0:31	RX Channel 6	
MAL0_RXCTP7R		0:31	RX Channel 7	
MAL0_RXCTP8R		0:31	RX Channel 8	Virtual Channel 8–15 EMAC1 RX (physical RX channel 1)
MAL0_RXCTP9R		0:31	RX Channel 9	
MAL0_RXCTP10R		0:31	RX Channel 10	
MAL0_RXCTP11R		0:31	RX Channel 11	
MAL0_RXCTP12R		0:31	RX Channel 12	
MAL0_RXCTP13R		0:31	RX Channel 13	
MAL0_RXCTP14R		0:31	RX Channel 14	
MAL0_RXCTP15R		0:31	RX Channel 15	

Table 27-5. MAL Register to Channel Mapping (Continued)

Register	Field	Bit(s)	Channel	Comment
MAL0_RXCTP16R		0:31	RX Channel 16	Virtual Channel 16–23 EMAC2 RX (physical RX channel 2)
MAL0_RXCTP17R		0:31	RX Channel 17	
MAL0_RXCTP18R		0:31	RX Channel 18	
MAL0_RXCTP19R		0:31	RX Channel 19	
MAL0_RXCTP20R		0:31	RX Channel 20	
MAL0_RXCTP21R		0:31	RX Channel 21	
MAL0_RXCTP22R		0:31	RX Channel 22	
MAL0_RXCTP23R		0:31	RX Channel 23	
MAL0_RXCTP24R		0:31	RX Channel 24	Virtual Channel 24–31 EMAC3 RX (physical RX channel 3)
MAL0_RXCTP25R		0:31	RX Channel 25	
MAL0_RXCTP26R		0:31	RX Channel 26	
MAL0_RXCTP27R		0:31	RX Channel 27	
MAL0_RXCTP28R		0:31	RX Channel 28	
MAL0_RXCTP29R		0:31	RX Channel 29	
MAL0_RXCTP30R		0:31	RX Channel 30	
MAL0_RXCTP31R		0:31	RX Channel 31	
MAL0_RCBS0	RCBS	24:31	RX Channel 0	Virtual Channel 0–7 EMAC0 RX (physical RX channel 0)
MAL0_RCBS1	RCBS	24:31	RX Channel 1	
MAL0_RCBS2	RCBS	24:31	RX Channel 2	
MAL0_RCBS3	RCBS	24:31	RX Channel 3	
MAL0_RCBS4	RCBS	24:31	RX Channel 4	
MAL0_RCBS5	RCBS	24:31	RX Channel 5	
MAL0_RCBS6	RCBS	24:31	RX Channel 6	
MAL0_RCBS7	RCBS	24:31	RX Channel 7	
MAL0_RCBS8	RCBS	24:31	RX Channel 8	Virtual Channel 8–15 EMAC1 RX (physical RX channel 1)
MAL0_RCBS9	RCBS	24:31	RX Channel 9	
MAL0_RCBS10	RCBS	24:31	RX Channel 10	
MAL0_RCBS11	RCBS	24:31	RX Channel 11	
MAL0_RCBS12	RCBS	24:31	RX Channel 12	
MAL0_RCBS13	RCBS	24:31	RX Channel 13	
MAL0_RCBS14	RCBS	24:31	RX Channel 14	
MAL0_RCBS15	RCBS	24:31	RX Channel 15	

Table 27-5. MAL Register to Channel Mapping (Continued)

Register	Field	Bit(s)	Channel	Comment
MAL0_RCBS16	RCBS	24:31	RX Channel 16	Virtual Channel 16–23 EMAC2 RX (physical RX channel 2)
MAL0_RCBS17	RCBS	24:31	RX Channel 17	
MAL0_RCBS18	RCBS	24:31	RX Channel 18	
MAL0_RCBS19	RCBS	24:31	RX Channel 19	
MAL0_RCBS20	RCBS	24:31	RX Channel 20	
MAL0_RCBS21	RCBS	24:31	RX Channel 21	
MAL0_RCBS22	RCBS	24:31	RX Channel 22	
MAL0_RCBS23	RCBS	24:31	RX Channel 23	
MAL0_RCBS24	RCBS	24:31	RX Channel 24	Virtual Channel 24–31 EMAC3 RX (physical RX channel 3)
MAL0_RCBS25	RCBS	24:31	RX Channel 25	
MAL0_RCBS26	RCBS	24:31	RX Channel 26	
MAL0_RCBS27	RCBS	24:31	RX Channel 27	
MAL0_RCBS28	RCBS	24:31	RX Channel 28	
MAL0_RCBS29	RCBS	24:31	RX Channel 29	
MAL0_RCBS30	RCBS	24:31	RX Channel 30	
MAL0_RCBS31	RCBS	24:31	RX Channel 31	

27.9.2 MAL Configuration Register (MAL0_CFG)

This register defines the operational mode of MAL. Unless a configuration change is required during system operation, the configuration register needs to be set only during system initialization.

Figure 27-9. MAL Configuration Register (MAL0_CFG)

0	SR	MAL Software Reset 0 MAL reset is complete 1 Reset the MAL	Generates a general reset to MAL through a software command. After setting this bit, MAL hardware (registers, interface and internal state machines) returns to the power-on reset value. The software writes 1 to this bit in order to drive MAL to the reset state. The bit is cleared by the hardware when the reset is completed (one system clock).
1:7		Reserved	
8:9	RPP	Read PLB Priority 00 Lowest 01 10 11 Highest	Determines the priority of MAL requests on the PLB for read transactions.
10:11	RMBS	Read Max Burst Size 00 4 01 8 10 16 11 32 (recommended)	Maximum PLB burst size for read transactions. Determines the maximum data transfers (RdDAcks) per read burst transaction.

12:13	WPP	Write PLB Priority 00 Lowest 01 10 11 Highest	Determines the priority of MAL requests on the PLB for write transactions.
14:15	WMBS	Write Max Burst Size 00 4 01 8 10 16 11 32 (recommended)	Maximum PLB burst size for write transactions. Determines the maximum data transfers (WrDAcks) per write burst transaction.
16	PLBLE	PLB Lock Error 0 LOCKERROR signal not applied to the PLB slave 1 LOCKERROR signal applied to the PLB slave	When this bit is set, MAL applies the LOCKERROR signal to the PLB slave when it is the initiator during PLB transactions.
17	PLBB	PLB Burst 0 Burst transactions not allowed 1 Burst transactions allowed	When this bit is reset, MAL is not allowed to perform burst transactions.
18:23		Reserved	
24	OPBBL	OPB Bus Lock 0 OPB not locked 1 OPB locked	When this bit is set, MAL locks the OPB during data transfers to and from the EMACs.
25:28		Reserved	
29	EOPIE	End of Packet Interrupt Enable 0 Generate interrupt on every end-of-packet only if the buffers I bit is set 1 Generate interrupt is on every end-of-packet	When this bit is set, an interrupt is generated on every end of packet (both transmit and receive). When clear, end of packet/buffer interrupt is generated only if the buffers I bit is set (1). **Note:** An interrupt is generated for every descriptor on which the I bit is set, regardless of the state of the EOPIE bit.
30	LEA	Locked Error Active 0 Handle errors in a non-locked mode 1 Handle errors in locked mode	Determines the MAL error handling mode. When this bit is set, MAL will handle errors in the locked mode. Otherwise, it will handle errors in a non-locked mode.
31	SD	MAL Scroll Descriptor 0 Do not scroll to the first descriptor of the next packet 1 Scroll to the first descriptor of the next packet	Determines whether or not MAL should scroll to the first descriptor of the next packet, following an early packet termination initiated by the related EMAC. When set, Scrolling mode is active.

27.9.3 Channel Active Set and Reset Registers (MAL0_TXCASR, TXCARR, RXCASR, RXCASR

For the Channel Active Set/Reset Registers, each bit represents its associated channel (bit 0 for channel 0, and so on). When a bit is equal to 1, the channel has been enabled for operation. When a bit is equal to 0, the channel is disabled (MAL ignores any requests for service). If a channel is active when its enable bit is cleared, MAL stops processing the current packet. After the channel's enable bit is cleared, MAL goes back to the top of the channel descriptor table (pointed to by MAL0_TXCPTxR, MAL0_RXCPTxR).

- To enable a channel:
  - Write a 1 to its corresponding bit in the Channel Active Set Register (CASR).
  - Multiple channels can be enabled with a single CASR register write.
- To stop and reset a channel:
  - Write a 1 to its corresponding bit in the Channel Active Reset Register (CARR).
  - Writing a 0 to bits in the CARR registers has no effect on the channels.
  - Multiple channels can be reset with a single CARR register write.

MAL also clears the enable bit of a channel following an indication of an error on the channel. The CASR or CARR register(s) can be read to determine which channels are currently active. The following figures describe these registers.

Note: For channel assignment see *Section 27.9.1 MAL Register to Channel Mapping* on page 974

Figure 27-10. Transmit Channel Active Set Register (MAL0_TXCASR)

0:1	TCAS01	Transmit Channel 0:1 Active Set Bit 0 TX Channel 0 (EMAC0) Bit 1 TX Channel 1 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). When 1 is written to the bit, channel operation is enabled.
2:3	TCAS23	Transmit Channel 2:3 Active Set (PPC460GT only) Bit 2 TX Channel 2 (EMAC2) Bit 3 TX Channel 3 (EMAC3)	Each bit represents its related channel (bit 2 for channel 2, and so on). When 1 is written to the bit, channel operation is enabled.
4:31		Reserved	

Figure 27-11. Transmit Channel Active Reset Register (MAL0_TXCARR)

0:1	TCAR01	Transmit Channel 0:1 Active Reset Bit 0 TX Channel 0 (EMAC0) Bit 1 TX Channel 1 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). When 1 is written to the bit, channel operation is disabled.
2:3	TCAR23	Transmit Channel 2:3 Active Reset (PPC460GT only) Bit 2 TX Channel 2 (EMAC2) Bit 3 TX Channel 3 (EMAC3)	Each bit represents its related channel (bit 2 for channel 2, and so on). When 1 is written to the bit, channel operation is disabled.
4:31		Reserved	

Figure 27-12. Receive Channel Active Set Register (MAL0_RXCASR)

0:1	RCAS01	Receive Channel 0:1 Active Set Bit 0 RX Physical Channel 0 (EMAC0) Bit 1 RX Physical Channel 1 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). When 1 is written to the bit, channel operation is enabled.
2:3	RCAS23	Receive Channel 2:3 Active Set (PPC460GT only) Bit 2 RX Physical Channel 2 (EMAC2) Bit 3 RX Physical Channel 3 (EMAC3)	Each bit represents its related channel (bit 2 for channel 2, and so on). When 1 is written to the bit, channel operation is enabled.
4:31		Reserved	

Figure 27-13. Receive Channel Active Reset Register (MAL0_RXCARR)

0:1	RCAR01	Receive Channel 0:1 Active Reset Bit 0 RX Physical Channel 0 (EMAC0) Bit 1 RX Physical Channel 1 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). When 1 is written to the bit, channel operation is disabled.
2:3	RCAR23	Receive Channel 2:3 Active Reset (PPC460GT only) Bit 2 RX Physical Channel 2 (EMAC2) Bit 3 RX Physical Channel 3 (EMAC3)	Each bit represents its related channel (bit 0 for channel 0, and so on). When 1 is written to the bit, channel operation is disabled.
4:31		Reserved	

27.9.4 End of Buffer Interrupt Status Registers (MAL0_TXEOBISR, MAL0_RXEOBISR)

Each bit in the transmit End-of-Buffer Interrupt Status and receive End-of-Buffer Interrupt Status registers is related to a channel's buffer descriptor table.

The transmit End-of-Buffer Interrupt Status register contains the End-of-Buffer Status bits for each transmit channel. The receive End-of-Buffer Interrupt Status register contains the End-of-Buffer Status bits for the receive channels. The mechanism (as described below) for both receive and transmit registers is the same.

MAL sets a channel's bit in one of the following conditions:

- When MAL finishes the processing of a buffer (writes back the status to the current descriptor), the related bit in this register is set if the I bit in the descriptor status is set.
- When MAL finishes the processing of a packet (writes back the status of the packet's last buffer) and MAL0_MCR[EOPIE] is set.

  Note: In case MAL finishes the processing of a packet which is backed up, MAL doesn't consider it as an end of packet. Therefore, MAL will not set the appropriate channel bit in the End-of-Buffer Register.
- When the Bad Packet bit is set in the EMAC channel Status halfword.

The device driver resets the interrupt by writing a 1 to the related bit. Writing a 0 has no effect.

Figure 27-14. Transmit End of Buffer Interrupt Status Register (MAL0_TXEOBISR)

0:1	TCEI01	Transmit Channel 0:1 End-of-Buffer Interrupt Bit 0 TX Channel 0 (EMAC0) Bit 1 TX Channel 1 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). Writing 1 to a bit clears it.
2:3	TCEI23	Transmit Channel 2:3 End-of-Buffer Interrupt (PPC460GT only) Bit 2 TX Channel 2 (EMAC2) Bit 3 TX Channel 3 (EMAC3)	Each bit represents its related channel (bit 2 for channel 2, and so on). Writing 1 to a bit clears it.
4:31		Reserved	

Note: In the following register, the PPC460EX/EXr has 16 virtual receive channels and uses the first 16 bits while the remaining 16 are reserved. The PPC460GT has 32 virtual receive channels and uses all 32 bit of the register. For channel assignment see *Section 27.9.1 MAL Register to Channel Mapping* on page 974

Figure 27-15. Receive End of Buffer Interrupt Status Register (MAL0_RXEOBISR)			
0:15	RCEI16	Receive Channel End-of-Buffer Interrupt Sixteen (16) virtual receive channels (PPC460EX/EXr and PPC460GT). Bit 0 Virtual RX Channel 0 (EMAC0) Bit 1 Virtual RX Channel 1 (EMAC0) Bit 2 Virtual RX Channel 2 (EMAC0) Bit 3 Virtual RX Channel 3 (EMAC0) Bit 4 Virtual RX Channel 4 (EMAC0) Bit 5 Virtual RX Channel 5 (EMAC0) Bit 6 Virtual RX Channel 6 (EMAC0) Bit 7 Virtual RX Channel 7 (EMAC0) Bit 8 Virtual RX Channel 8 (EMAC1) Bit 9 Virtual RX Channel 9 (EMAC1) Bit 10 Virtual RX Channel 10 (EMAC1) Bit 11 Virtual RX Channel 11 (EMAC1) Bit 12 Virtual RX Channel 12 (EMAC1) Bit 13 Virtual RX Channel 13 (EMAC1) Bit 14 Virtual RX Channel 14 (EMAC1) Bit 15 Virtual RX Channel 15 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). Writing 1 to a bit clears it.
16:31	RCEI32	Receive Channel End-of-Buffer Interrupt (PPC460GT only) (The PPC460GT has 16 additional virtual receive channels.) Bit 16 Virtual RX Channel 16 (EMAC2) Bit 17 Virtual RX Channel 17 (EMAC2) Bit 18 Virtual RX Channel 18 (EMAC2) Bit 19 Virtual RX Channel 19 (EMAC2) Bit 20 Virtual RX Channel 20 (EMAC2) Bit 21 Virtual RX Channel 21 (EMAC2) Bit 22 Virtual RX Channel 22 (EMAC2) Bit 23 Virtual RX Channel 23 (EMAC2) Bit 24 Virtual RX Channel 24 (EMAC3) Bit 25 Virtual RX Channel 25 (EMAC3) Bit 26 Virtual RX Channel 26 (EMAC3) Bit 27 Virtual RX Channel 27 (EMAC3) Bit 28 Virtual RX Channel 28 (EMAC3) Bit 29 Virtual RX Channel 29 (EMAC3) Bit 30 Virtual RX Channel 30 (EMAC3) Bit 31 Virtual RX Channel 31 (EMAC3)	Each bit represents its related channel (bit 16 for channel 16, and so on). Writing 1 to a bit clears it.

27.10 Error Registers

The following paragraphs describe MAL error registers. For more information about MAL errors see *MAL Error Handling* on page 965.

27.10.1 MAL Error Status Register (MAL0_ESR)

This register holds the information about the error that occurred and the interrupts status. The register includes the following fields:

- **Error status bits** – This field holds the error information. The information includes the number of the channel on which the error occurred (if known) and the type of the error. The error can be either the last detected error

or a locked error if "Locked error mode" is active. (See *Operational Error Modes* on page 967 for description of the Locked error mode.)

The error status field includes an "Error Valid" bit which indicates whether there is error information in the error status field or not. The error status bits are not valid when the "Error Valid" bit is cleared (by writing 1 to this bit).

- **Interrupt status bits** – Every error detected by MAL sets a related bit in the interrupt status field. The interrupt status bits may be cleared by software by writing 1 to the bit to be cleared. The bits in this field are accumulative (more than one interrupt may be indicated here). These bits are masked by the IER (Interrupt Enable Register) to create a maskable interrupt (which is the MAL_SERR interrupt in the UIC).

Note: In order to reset the interrupt bits and the Error valid bit in the Error Status register, 1 must be written to the related bit. Writing 0 has no effect.

More than one bit can be cleared at a time and only R/W bits can be reset.

Figure 27-16. MAL Error Status Register (MAL0_ESR)

0	EVB	Error Valid Bit 0 Bit 1:15 are available for latching new error information. 1 Bits 1:15 contain last error. A new error cannot be latched.	When this bit is set, bits 1-6 include the ID of the erroneous channel (in case of OPB errors). Bits 11–15 indicate the type of error. In non-locked mode, the error indication describes the last error that had occurred. In locked mode, the error is the first one that had occurred after this bit was cleared. This bit is set when an error occurs and remains set until reset by the software. In locked mode, new errors cannot be latched in the error lock indication fields if this bit is set
1	CIDT	Channel ID Type 0 CID field represents TX channel 1 CID field represents RX channel	Identifies current error as an RX channel error or TX channel error.
2:6	CID	Channel ID Number	Indicates the number of the channel that caused the error. See *Figure 27-5 MAL Register to Channel Mapping*. **Note:** An error on the PLB cannot be related to a channel. The error condition may be resolved by using the error information optionally locked in the PLB slave.
7		Reserved	
8	PTE	PLB Timeout Error 0 No error 1 Error has occurred	Indicates the error is a PLB timeout
9	PRE	PLB Read Error 0 No error 1 Error has occurred	
10	PWE	PLB Write Error 0 No error 1 Error has occurred	
11	DE	Descriptor Error 0 No error 1 Non-valid descriptor	This error indicates that an invalid descriptor was accessed. For a TX error, the invalid descriptor is not the first decriptor for a TX packet. For a RX error, the invalid descriptor is any descriptor for a RX packet.
12		Reserved	

13	OTE	OPB Timeout Error 0 No error 1 Error has occurred	Indicates the error is an OPB timeout.
14	OSE	OPB Slave Error 0 No error 1 Error occurred	Indicates the error is an error indication asserted by an OPB slave.
15	PEIN	PLB Bus Error Indication 0 No error 1 Error occurred	When this bit is set, the detected error is a PLB error. There is no meaning to the Channel ID field in this case.
16:23		Reserved	
24	PTEI	PLB Timeout Error Interrupt 0 No error 1 Error occurred	This bit is set following a PLB timeout error indication. Set condition for this bit generates a maskable interrupt.
25	PREI	PLB Read Error Interrupt 0 No error 1 Error occurred	This bit is set following a PLB read error indication. Set condition for this bit generates a maskable interrupt.
26	PWEI	PLB Write Error Interrupt 0 No error 1 Error occurred	This bit is set following a PLB write error indication. Set condition for this bit generates a maskable interrupt.
27	DEI	Descriptor Error Interrupt 0 No error 1 Descriptor data error recognized	A descriptor data error is recognized during an access to the descriptor table. This interrupt indicates that an invalid descriptor was accessed. For a TX error, the invalid descriptor is not the first decriptor for a TX packet. For a RX error, the invalid descriptor is any descriptor for a RX packet. Set condition for this bit generates a maskable interrupt.
28		Reserved	
29	OTEI	OPB Timeout Error Interrupt 0 No error 1 OPB time-out	This bit is set following an OPB time out error indication. Set condition for this bit generates a maskable interrupt.
30	OSEI	OPB Slave Error Interrupt 0 No error 1 OPB error from a slave	This bit is set following an OPB error indicated by the slave. Set condition for this bit generates a maskable interrupt.
31	PBEI	PLB Bus Error Interrupt 0 No error 1 PLB error indication	This bit is set following a PLB error indication (from the PLB slave). Set condition for this bit generates a maskable interrupt.

27.10.2 MAL Interrupt Enable Register (MAL0_IER)

Each bit in the following register, when it is set, enables assertion of the MAL0 SERR interrupt in the UIC when the related bit in the MAL0_ESR (interrupt bit) is set.

Figure 27-17. MAL Interrupt Enable Register (MAL0_IER)			
0:23		Reserved	
24	PT	PLB Timeout Interrupt 0 Disabled 1 Enabled	
25	PRE	PLB Read Error Interrupt 0 Disabled 1 Enabled	
26	PWE	PLB Write Error Interrupt 0 Disabled 1 Enabled	
27	DE	Descriptor Error Interrupt 0 Disabled 1 Enabled	
28		Reserved	
29	OTE	OPB Timeout Error Interrupt 0 Disabled 1 Enabled	
30	OE	OPB Slave Error Interrupt 0 Disabled 0 Enabled	
31	PE	PLB Error Interrupt 0 Disabled 1 Enabled	

27.10.3 Descriptor Error Interrupt Registers (MAL0_TXDEIR, MAL0_RXDEIR)

Each bit in the following registers is related to a channel buffer descriptor table. Each bit indicates a descriptor data error related to a certain channel.

The Transmit Descriptor Error Interrupt register contains the descriptor errors bits of the transmit channels. The Receive Descriptor Error Interrupt register contains the descriptor errors bits of the receive channels. The mechanism (as described below) for both receive and transmit registers is the same.

MAL sets a channel's bit when a descriptor data error was recognized during access to the descriptor table of a specific channel (see *MAL Error Causes* on page 965).

The device driver resets the interrupt by writing a 1 to the related bit. Writing a 0 has no effect. When one or more of the Transmit Descriptor Error Interrupt bits is set, then the MAL_TX_DESC_ERR_INT bit is set. When one or more of the Transmit Descriptor Error Interrupt bits is set, the MAL TXDE interrupt in the UIC is set. When one or more of the Receive Descriptor Error Interrupt bits is set, the MAL RXDE interrupt in the UIC is set.

Figure 27-18. Transmit Descriptor Error Interrupt Register (MAL0_TXDEIR)			
0:1	TXDE01	Transmit Descriptor Error Interrupt Bit 0 TX Channel 0 (EMAC0) Bit 1 TX Channel 1 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). When one or more bits are set to 1, MAL TXDE interrupt is set. Writing 1 to a bit clears it.

2:3	TXDE23	Transmit Descriptor Error Interrupt (PPC460GT only) Bit 2 TX Channel 2 (EMAC2) Bit 3 TX Channel 3 (EMAC3)	Each bit represents its related channel (bit 2 for channel 2, and so on). When one or more bits are set to 1, MAL TXDE interrupt is set. Writing 1 to a bit clears it.
4:31		Reserved	

Note: For channel assignment see section *MAL Register to Channel Mapping* on page 974

Figure 27-19. Receive Descriptor Error Interrupt Register (MAL0_RXDEIR)

0:15	RXDE16	Receive Descriptor Error Interrupt Sixteen (16) virtual receive channels (PPC460EX/EXr and PPC460GT). Bit 0 Virtual RX Channel 0 (EMAC0) Bit 1 Virtual RX Channel 1 (EMAC0) Bit 2 Virtual RX Channel 2 (EMAC0) Bit 3 Virtual RX Channel 3 (EMAC0) Bit 4 Virtual RX Channel 4 (EMAC0) Bit 5 Virtual RX Channel 5 (EMAC0) Bit 6 Virtual RX Channel 6 (EMAC0) Bit 7 Virtual RX Channel 7 (EMAC0) Bit 8 Virtual RX Channel 8 (EMAC1) Bit 9 Virtual RX Channel 9 (EMAC1) Bit 10 Virtual RX Channel 10 (EMAC1) Bit 11 Virtual RX Channel 11 (EMAC1) Bit 12 Virtual RX Channel 12 (EMAC1) Bit 13 Virtual RX Channel 13 (EMAC1) Bit 14 Virtual RX Channel 14 (EMAC1) Bit 15 Virtual RX Channel 15 (EMAC1)	Each bit represents its related channel (bit 0 for channel 0, and so on). Writing 1 to a bit clears it.
16:31	RXDE32	Receive Descriptor Error Interrupt (PPC460GT only) (The PPC460GT has 16 additional virtual receive channels.) Bit 16 Virtual RX Channel 16 (EMAC2) Bit 17 Virtual RX Channel 17 (EMAC2) Bit 18 Virtual RX Channel 18 (EMAC2) Bit 19 Virtual RX Channel 19 (EMAC2) Bit 20 Virtual RX Channel 20 (EMAC2) Bit 21 Virtual RX Channel 21 (EMAC2) Bit 22 Virtual RX Channel 22 (EMAC2) Bit 23 Virtual RX Channel 23 (EMAC2) Bit 24 Virtual RX Channel 24 (EMAC3) Bit 25 Virtual RX Channel 25 (EMAC3) Bit 26 Virtual RX Channel 26 (EMAC3) Bit 27 Virtual RX Channel 27 (EMAC3) Bit 28 Virtual RX Channel 28 (EMAC3) Bit 29 Virtual RX Channel 29 (EMAC3) Bit 30 Virtual RX Channel 30 (EMAC3) Bit 31 Virtual RX Channel 31 (EMAC3)	Each bit represents its related channel (bit 16 for channel 16, and so on). Writing 1 to a bit clears it.

27.10.4 Descriptor Base Address Registers (MAL0_TXBADDR, MAL0_RXBADDR)

Descriptor base address registers are used to update and read to/from all channels. MAL determines which channel should be used according to the specific transmit/receive CTPR used. The flow of programming a table pointer address of 36 bits is described in the paragraphs that follow. Repeat the following steps for each channel.

Writing Flow

1. Write the four addresses msbs to the appropriate base address register (MAL0_TXBADDR or MAL0_RXBADDR).
2. Write the 32 least significant bits to the specific channel's address register (MAL0_TXCTPxR or MAL0_RXCTPxR)
3. MAL stores the full 36-bit address combined from the above two DCR accesses in its internal memory.

Reading Flow

1. Read the 32 least significant bits of the specific channel's address register.
2. MAL updates the (MAL0_TXBADDR or MAL0_RXBADDR) base address register according to the channel read in phase 1.
3. Read the 4 addresses' msbs from bits 28 to 31 in the common base address register.

Figure 27-20. Transmit Descriptor Base Address Register (MAL0_TXBADDR)

0:27		Reserved	
28:31	DBA	Descriptor Base Address	

Figure 27-21. Receive Descriptor Base Address Register (MAL0_RXBADDR)

0:27		Reserved	
28:31	DBA	Descriptor Base Address	

27.10.5 Channel Table Pointer Registers (MAL0_TXCTPxR, MAL0_RXCTPxR)

MAL uses Receive Channel Table Pointer Registers, one for each receive channel, and Transmit Channel Table Pointer Registers, one for each transmit channel. The Channel Table Pointer Registers point to the base address, in memory, of the buffer descriptor table used by each channel.

Note 1: When changing the value of any of the MAL0_TXCTPxR registers, the changed transmit channels must be idle. To verify a channel is idle, check the device's Transmit Idle bit. Another way to assure that the channels are idle is to disable the channels before changing the MAL0_TXCTPxR register, and then re-enable them once the MAL0_TXCTPxR register is set to its new value.

Note 2: Although MAL supports a full 36-bit address space, no data buffer is allowed to cross a 32-bit address space border, and no buffer descriptor table is allowed to cross a 32-bit address space border.

Each channel has a full and separate 36-bit address space with the following limitations:

1. No data buffer is allowed to cross a 32-bit address space border.
2. No buffer descriptor table is allowed to cross a 32-bit address space border.

As the DCR bus is only 32 bits wide, each channel's table pointer needs to be configured in two DCR accesses as follows:

- Configure the 4 msbs using a common register (MAL0_TXBADDR or MAL0_RXBADDR)
- Configure the 32 msbs using a channel private register (MAL0_TXCTPxR or MAL0_RXCTPxR)

The Transmit and Receive Channel Table Pointer Registers have an identical format, as shown in *Figure 27-22* and *Figure 27-23*. MAL0 has two (PPC460EX/EXr) or four (PPC460GT) Transmit Channel Table Pointer Registers and 16 (PPC460EX/EXr) or 32 (PPC460GT) Virtual Receive Channel Table Pointer Registers.

Note: The PPC460EX/EXr has two transmit channels and the PPC460GT has four transmit channels.

Figure 27-22. Transmit Channel Table Pointer Register (MAL0_TXCTPxR)

0:31		Channel Table Pointer	Pointer to the base address of the buffer descriptor table used by the channel. The value entered should point to a location in memory accommodating an aligned doubleword (the three least significant bits of the pointer must be 000).

Note: The PPC460EX/EXr has 16 virtual receive channels and the PPC460GT has 32 virtual receive channels. For channel assignments see *Section 27.9.1 MAL Register to Channel Mapping* on page 974.

Figure 27-23. Receive Channel Table Pointer Register (MAL0_RXCTPxR)

0:31		Channel Table Pointer	Pointer to the base address of the buffer descriptor table used by the channel. The value should point to a location in memory accommodating an aligned doubleword (the three least significant bits of the pointer must be 000).

The Channel Table Pointer registers retain their value following Soft Reset or Channel Reset.

27.10.6 RX Channel Buffer Size Register (MAL0_RCBSx)

Each receive channel has a fixed buffer length. The receive channel buffer size registers are configured by the device driver to determine the length of the buffer. This size is the maximal number of bytes that can be stored in the buffer. More than one buffer is used to store the incoming packet in case the length of the incoming packet data is more than the channel's buffer length (as defined by the channel's related register).

The buffer length for a given channel can be from 16 bytes to (4KB-16) bytes (in 16-byte steps). This length is represented by a 8-bit field. The buffer size in bytes is calculated as follows: RX_Buffer_Size_Field x 16.

Note: The PPC460EX/EXr has 16 virtual receive channels and the PPC460GT has 32 virtual receive channels. For channel assignments see *Section 27.9.1 MAL Register to Channel Mapping* on page 974.

Figure 27-24. RX Channel Buffer Size Register (MAL0_RCBSx)

0:23		Reserved	
24:31	RCBS	Receive Channel Buffer Size	

27.10.7 MAL Related System Device Control Registers - SDRs

The following four SDRs select burst lengths and size of the MAL receive and transmit buffers.

SDR0_MALxxx registers are accessed indirectly through the SDR0_CFGADDR and SDR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions. *Table 3-1* on page 154 lists the DCR addresses for the SDR0_CFGADDR and SDR0_CFGDATA registers.

To read or write one of the SDR0_MALxxx registers, software first writes the register address offset of the target register into the SDR0_CFGADDR register. The target register can then be read or written through the SDR0_CFGDATA DCR register. The following PowerPC code illustrates this procedure by reading the SDR0_MALRBL register:

```
li r3, SDR0_MALRBL_Offset
mtdcr SDR0_CFGADDR, r3      ! write SDR0_CFGADDR with the SDR0_MALRB DCR offset address
mfdcr r4, SDR0_CFGDATA      ! read the content of SDR0_MALRBL from SDR0_CFGDATA
```

Table 27-6. MAL SDRs

Mnemonic	Register	Offset	Access	Page
SDR0_MALRBL	MAL Receive Burst Length	0x02A0	R/W	989
SDR0_MALRBS	MAL Receive Bus Size	0x02E0	R/W	989
SDR0_MALTBL	MAL Transmit Burst Length	0x0280	R/W	990
SDR0_MALTBS	MAL Transmit Bus Size	0x02C0	R/W	990

27.10.7.1 MAL Receive Burst Length Register (SDR0_MALRBL)

Figure 27-25. MAL Receive Burst Length Register (SDR0_MALRBL)

0		Reserved	
1:3	RBL0	MAL Receive Burst Length 0 Requested amount of data to be received between MAL and EMAC0 in a single transaction.	Reset value = 0b101 = 64 words = 256 bytes. Defines the maximum length of data transferred in a burst on the EOPB
4		Reserved	
5:7	RBL1	MAL Receive Burst Length 1 Requested amount of data to be received between MAL and EMAC1 in a single transaction.	Reset default value = 0b101 (64 words). Defines the maximum length of data transferred in a burst on the EOPB.
8		Reserved	
9:11	RBL2	MAL Receive Burst Length 2 (PPC460GT only) Requested amount of data to be received between MAL and EMAC2 in a single transaction.	Reset default value = 0b101 (64 words). Defines the maximum length of data transferred in a burst on the EOPB.
12		Reserved	
13:15	RBL3	MAL Receive Burst Length 3 (PPC460GT only) Requested amount of data to be received between MAL and EMAC3 in a single transaction.	Reset default value = 0b101 (64 words). Defines the maximum length of data transferred in a burst on the EOPB.
16:31		Reserved	
Note: Use the default values for all the bits described in this register.			

27.10.7.2 MAL Receive Bus Size Register (SDR0_MALRBS)

Figure 27-26. MAL Receive Bus Size Register (SDR0_MALRBS)

0:3	RBS	MAL Receive Bus Size 0000 Reserved 1111 128-bit bus size	Reset value = 0b1111. Indicates the MAL EOPB interface to EMAC data-bus width for all EMAC channels.
4:31		Reserved	
Note: Use the default values for all the bits described in this register.			

27.10.7.3 MAL Transmit Burst Length Register (SDR0_MALTBL)

Figure 27-27. MAL Transmit Burst Length Register (SDR0_MALTBL)

0		Reserved	
1:3	TBL0	MAL Transfer Burst Length 0 Requested amount of data to be transferred between MAL and EMAC0 in a single transaction.	Defines the maximum burst length transferred between MAL and EMAC0. Reset default value = 0b101 (64 words).
4		Reserved	
5:7	TBL1	MAL Transfer Burst Length 1 Requested amount of data to be transferred between MAL and EMAC1 in a single transaction.	Defines the maximum burst length transferred between MAL and EMAC1. Reset default value = 0b101 (64 words).
8		Reserved	
9:11	TBL2	MAL Transfer Burst Length 2 (PPC460GT only) Requested amount of data to be transferred between MAL and EMAC2 in a single transaction.	Defines the maximum burst length transferred between MAL and EMAC2. Reset default value = 0b101 (64 words).
12		Reserved	
13:15	TBL3	MAL Transfer Burst Length 3 (PPC460GT only) Requested amount of data to be transferred between MAL and EMAC3 in a single transaction.	Defines the maximum burst length transferred between MAL and EMAC3. Reset default value = 0b101 (64 words).
8:31		Reserved	
Note: Use the default values for all of the bits described in this register.			

27.10.7.4 MAL Transmit Bus Size Register (SDR0_MALTBS)

Figure 27-28. MAL Transmit Bus Size Register (SDR0_MALTBS)

0:3	TBS	MAL Transfer Bus Size 0000 Reserved 1111 128-bit bus size	Reset value = 0b1111. Indicates the OPB slave interface width for all four EMAC channels is 128 bits
4:31		Reserved	
Note: Use the default values for all of the bits described in this register.			

27.11 Interrupt Coalescing

Interrupt coalescing allows users to tailor the behavior of the Ethernet controller with respect to frame interrupts. Transmit and receive frames are handled separately with similar mechanisms. When coalescing is enabled (that is, when the bits 6, 7, 8, or 9 of UIC2 are set) transmit or receive frame interrupts are asserted when a predefined number of frames are transmitted or received, or a predefined time has elapsed and at least one frame has been transmitted or received, whichever occurs first.

There are two mutually exclusive ways to assert the interrupt:

1. The frame count threshold of transmitted or received frames has been exceeded, or
2. The timer has expired and at least one frame has been transmitted or received

In this implementation, the TX and RX functions are identical and present for the eight channels of the MAL.

The interrupt generated is synchronous to the PLB clock and has a pulse width of one clock period.

27.11.1 Coalescing by Frame Counter Method

When consecutive packets arrive at the host processor, it is possible for the host to process interrupts only and not perform any other tasks. This can also lead to bandwidth congestion. By enabling coalescing, users can group frame interrupts, thus reducing the total number of interrupts to be processed. The frame threshold is defined in the interrupt coalescing configuration register. In a typical scenario, the threshold number can be anywhere in 1 to 255. Once the number of frames transmitted or received reaches the predefined threshold, an interrupt occurs, the counter is automatically reset, and counting continues while the interrupt is active. The counter can also be reset when a pre-defined timer threshold is reached.

27.11.2 Coalescing by Timer Threshold

There can be a situation where no interrupt is raised for a long time and frames are waiting for processing. In this case, the user can define a timer threshold beyond which interrupts are forced. In the interrupt coalescing configuration register, the transmit and receive timer fields are set with reference to a number of PLB clocks.

When the threshold time is reached and at least one frame has been transmitted or received, an interrupt is raised, the timer is automatically reset, and counting continues while the interrupt is active. The counter can also be reset when the frame count threshold is reached.

The timer can be accessed by software. Reading the timer clears it.

27.11.3 Flushing Mechanism

Both the frame and timer threshold counters can be cleared by software. Software can also, optionally, generate an interrupt if any frames were transmitted or received since the last time the frame counter was reset.

27.12 Interrupt Coalescence Registers

The following registers control interrupt coalescing.

SDR0_ICxxxx registers are accessed indirectly through the SDR0_CFGADDR and SDR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions. *Table 3-1* on page 154 lists the DCR addresses for the SDR0_CFGADDR and SDR0_CFGDATA registers.

To read or write one of the SDR0_ICxxxx registers, software first writes the register address offset of the target register into the SDR0_CFGADDR register. The target register can then be read or written through the SDR0_CFGDATA DCR register. The following PowerPC code illustrates this procedure by reading the SDR0_ICCRTX0 register:

```
li r3, SDR0_ICCRTX0_Offset
mtdcr SDR0_CFGADDR, r3        ! write SDR0_CFGADDR with the SDR0_MALRB DCR offset address
mfdcr r4, SDR0_CFGDATA        ! read the content of SDR0_ICCRTX0 from SDR0_CFGDATA
```

Note: Channels 2 and 3 only apply only to the PPC460GT.

.

Table 27-7. Interrupt Coalescence Registers

Mnemonic	Register	DCR Numbers	Access	Page
SDR0_ICCRTX0:3	Interrupt Coalescing Control Register Transmit Channels 0:3	0x4410–0x4413	R/W	992
SDR0_ICCRRX0:3	Interrupt Coalescing Control Register Receive Channels 0:3	0x4414–0x4417	R/W	992
SDR0_ICCTRTX0:3	Interrupt Coalescing Count Threshold Register Transmit Channels 0:3	0x4418–0x441B	R/W	993
SDR0_ICCTRRX0:3	Interrupt Coalescing Count Threshold Register Receive Channels 0:3	0x441C–0x441F	R/W	993
SDR0_ICTSRTX0:3	Interrupt Coalescing Timer Status Register Transmit Channels 0:3	0x4420–0x4423	Read Clear	993
SDR0_ICTSRRX0:3	Interrupt Coalescing Timer Status Register Receive Channels 0:3	0x4424–0x4427	Read Clear	993
SDR0_ICVCMASK	Interrupt Coalescing Virtual Channel Mask	0x4428	R/W	993

27.12.1 Interrupt Coalescing Control Transmit Register Channel n (SDR0_ICCRTXn)

Figure 27-29. Interrupt Coalescing Control Transmit Register Channel n (SDR0_ICCRTXn)

0:8	FTHR	Frame Count Threshold	Reset value of the field is 0b000000001. Frame count threshold for transmit side of MAL channel n.
9	FLUSH	Flush with no Interrupt	Reset value of the field is 0b0. Write 1 to clear the timer and frame counters without generating an interrupt, then write 0 to enable the counters.
10	FLUWI	Flush with Interrupt	Reset value of the field is 0b0. Write 1 to clear the timer and frame counters and generate an interrupt if at least one frame was transmitted, then write 0 to enable the counters.
11:31		Reserved	

27.12.2 Interrupt Coalescing Control Receive Register Channel n (SDR0_ICCRRXn)

Figure 27-30. Interrupt Coalescing Control Receive Register Channel n (SDR0_ICCRRXn)

0:8	FTHR	Frame Count Threshold	Reset value of the field is 0b000000001. Frame count threshold for receive side of MAL channel n.
9	FLUSH	Flush with no Interrupt	Reset value of the field is 0b0. Write 1 to clear the timer and frame counters without generating an interrupt, then write 0 to enable the counters.
10	FLUWI	Flush with Interrupt	Reset value of the field is 0b0. Write 1 to clear the timer and frame counters and generate an interrupt if at least one frame was transmitted, then write 0 to enable the counters.
11:31		Reserved	

27.12.3 Interrupt Coalescing Count Threshold Register Transmit Channel n (SDR0_ICCTRTXn)

Figure 27-31. Interrupt Coalescing Count Threshold Register Transmit Channel n (SDR0_ICTRTXn)

0:31		Transmit Timer Register	Configure the value of the timer counter for transmit side of channel n of the MAL. Setting the value to zero (default) disables the timer.

27.12.4 Interrupt Coalescing Count Threshold Register Receive Channel n (SDR0_ICCTRRXn)

Figure 27-32. Interrupt Coalescing Count Threshold Register Receive Channel n (SDR0_ICTRRXn)

0:31		Receive Timer Register	Configure the value of the timer counter for receive side of channel n of the MAL. Setting the value to zero (default) disables the timer.

27.12.5 Interrupt Coalescing Timer Status Register Transmit Channel n (SDR0_ICTSRTXn)

Figure 27-33. Interrupt Coalescing Timer Status Register Transmit Channel n (SDR0_ICSRTXn)

0:31		Transmit Timer Status Register	Read and reset the timer counter of the transmit side of channel n of the MAL.

27.12.6 Interrupt Coalescing Timer Status Register Receive Channel n (SDR0_ICTSRRXn)

Figure 27-34. Interrupt Coalescing Timer Status Register Receive Channel n (SDR0_ICSRRXn)

0:31		Receive Timer Status Register	Read and reset the timer counter of the receive side of channel n of the MAL.

27.12.7 Interrupt Coalescing Virtual Channel Mask Register (SDR0_ICVCMASK)

Figure 27-35. Interrupt Coalescing Virtual Channel Mask Register (SDR0_ICVCMASK)

0:7	RQM0	Receive QoS Mask 0	Virtual Channel Mask corresponding to EMAC0. RQM0[0:7] masks MAL Receive EOB Interrupts 0–7 from being coalesced for Receive Channel 0.
8:15	RQM1	Receive QoS Mask 1	Virtual Channel Mask corresponding to EMAC1. RQM1[0:7] masks MAL Receive EOB Interrupts 8–15 from being coalesced for Receive Channel 1.
16:31		Reserved	

28. Ethernet Media Access Controller

The PPC460EX/EXr contains two Ethernet media access controllers (EMACs) and the PPC460GT contains four EMACs. The EMACs are generic implementations of the Ethernet Media Access Control (MAC) protocol complying with ANSI/IEEE Standard 802.3 and IEEE 802.3u supplement.

Each EMAC supports half-duplex (CSMA/CD), full-duplex operation for 10 Mbps and 100 Mbps operations, and operation in the 1000 Mbps range (Gigabit Ethernet) complying with IEEE 802.3z. In the Gigabit Ethernet mode, the real bandwidth is limited only by system latency (attributable to memory access layer (MAL), memory, arbitration, etc.).

Each of the EMACs are implemented identically, with the exception of register addresses. Because the EMACs operate identically, the rest of this chapter describes a single EMAC. The EMACs are referred to as EMAC0 and EMAC1. Except in the register summary tables in *EMAC Registers* on page 1019, the EMAC registers are prefixed "EMACx_" to denote the identical implementation of registers in each EMAC.

Each EMAC provides one on-chip peripheral bus (OPB based 32-bit bidirectional) slave interfaces and two other extended OPB (EOPB based 128-bit bidirectional) slave interfaces.The 32-bit OPB slave interface provides OPB masters (such as the CPU) access to the EMAC configuration and status registers by means of the PLB/OPB bridge. This enables the processor to read and write these registers. The two 128-bit EOPB slave interfaces are used to transfer packet data between the EMACs and the memory access layer (MAL) core, one for transmit data and the other for receive data. The MAL is the EOPB master. The EMAC supports up to 64 word long (256 bytes) burst-transfers on the EOPB interface.

The MAL is a specialized, dedicated, multi-channel DMA that resides between frame-based communication controllers (such as EMAC) and PLB accessible memory (such as DRAM or SRAM). The MAL transfers per-buffer EMAC configuration data, packet buffer data, and transfer-completion status data between a EMAC and memory. It provides separate channels for each of the EMAC's two MAL service request channels (one receive and one transmit). Software, such as a device driver, maintains a circular queue of buffers in external memory for each MAL channel, and manages the MAL's transfer of packet data between the data buffers and the networking protocol stack software. The MAL performs arbitration between EMAC channel service requests and processes the next buffer in the winning requestor's buffer descriptor table. After the transfer of data to or from the EMAC is completed, the MAL updates the descriptor's status field with the EMAC's completion status.

In the 10/100 Mbps range each EMAC uses a media independent interface (MII) to communicate with the Z media independent interface ZMII bridge or the RGMII bridge. The ZMII and RGMII bridges, described in detail in *EMAC to PHY Interface Bridges* on page 1055, communicate with standard physical devices (PHYs).

In the Gigabit Ethernet mode, the EMAC uses Gigabit MII (GMII) interface to communicate with the RGMII Bridge or ten bit interface (TBI) to communicate the serial GMII, (SGMII) SerDes. The EMAC implements a Gigabit Ethernet PHY coding sublayer (GPCS) needed for TBI used for SGMII. The RGMII Bridge is used for GMII and RGMII (see *EMAC to PHY Interface Bridges* on page 1055 for more details).

The EMAC has two independent receive and transmit FIFOs. Programmable MAL transfer-service request thresholds allow system specific adjustment to minimize receive overflows, transmit underruns and latency. Optionally, automatic sending of IEEE 802.3x Pause Packets for hardware implemented flow control is also triggered by the service thresholds.

As part of the remote monitoring (RMON) and management information base (MIB) defined in IEEE 802.3z, the EMAC contains registers that count the number of octets transmitted and received.

EMAC supports Jumbo frames, those with data fields sized over 1500 bytes, up to 9018 bytes when Virtual LAN (VLAN) tagging is not used, and 9022 bytes when VLAN tags are used. In addition to enabling the EMAC to transmit and receive Jumbo packets by setting EMACx_MR1[JPSM] = 1, the MAL device driver software must support packet reception and transmission by using multiple data buffers. The MAL supports a maximum data

buffer size of 4080 bytes. By using the RX buffer descriptor's status/control field flags First and Last, software is able to determine the beginning, middle, and end of the buffers composing a frame. For example, if a 9000 bytes packet is received, and the receive buffer size (RCBSx) is set to the maximum (0x000000FF), the first buffer descriptor will have its First flag set, the second buffer descriptor will have both the First and Last flags cleared, and the third descriptor will have the Last flag set. The device driver may be required to copy the data in these RX buffers into a single contiguous block of memory before passing the packet to the IP stack. Preferably, an interface that supports sending a link list of RX buffers to the IP stack can be supported.

28.1 EMAC Features

EMAC features:

- Triple speed (10/100/1000 Mbps) CSMA/CD (half-duplex) and full-duplex Ethernet MAC complying with ANSI/IEEE Std. 802.3, 802.3z and IEEE 802.3u supplement.
- Two 128-bit extended OPB (EOPB) interfaces to MAL with one receive and one transmit channel enable high throughput and performance which allows the interfaces to be connected to different MALs in one system via the same or separate MAL-EOPB buses.
- EMAC TX channel has a multiple status mechanism which allows EMAC to send the status/error word that is relevant to the last TX packet only after the next TX packet.
- Support for next page feature in the PCS auto-negotiation process.
- Automatic source address insertion or replacement for transmitted packets is a programmable option.
- Automatic stripping of frame padding bytes and frame check sequence (FCS) is a programmable option.

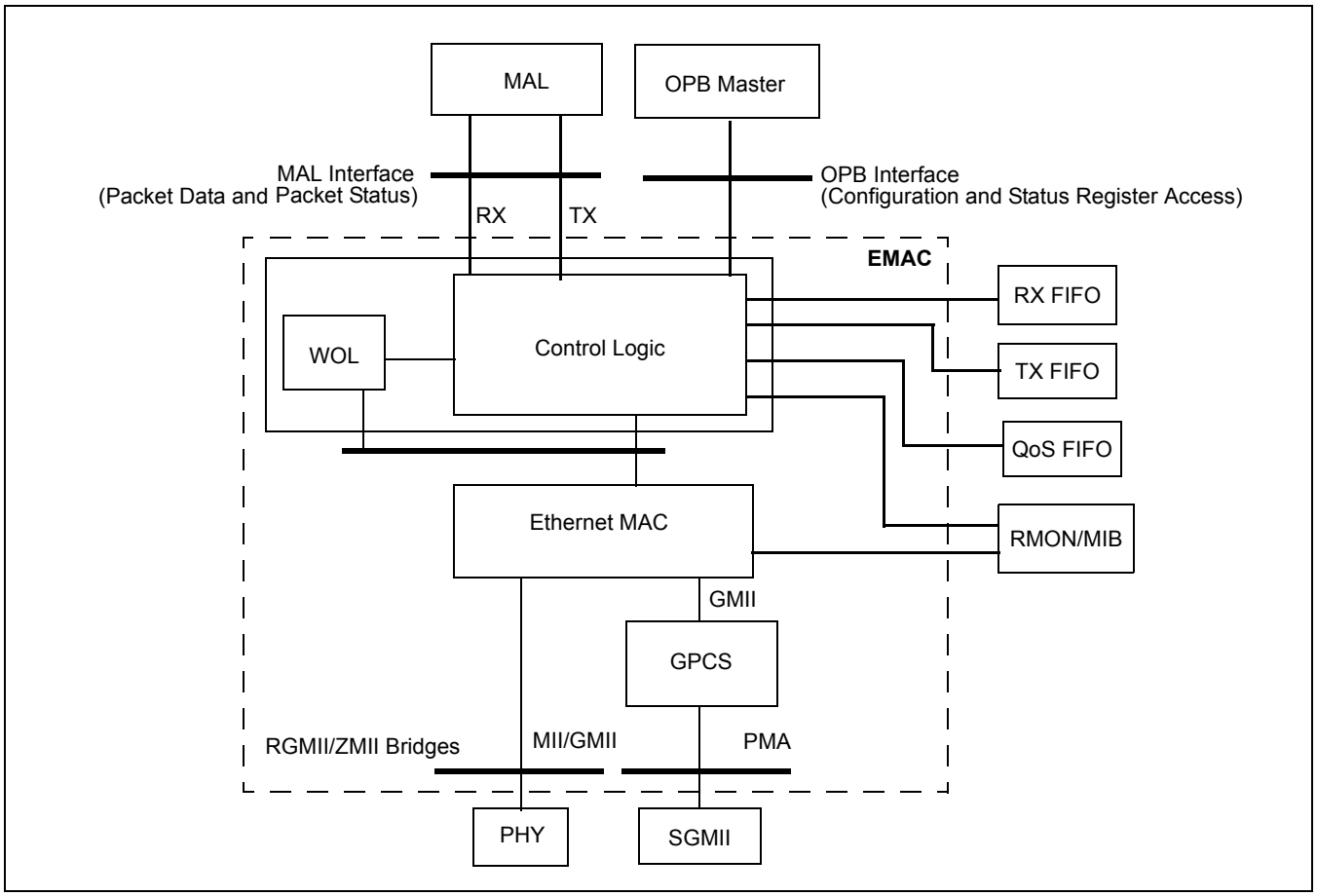
  When padding bytes are stripped, the padding and FCS field are removed. FCS stripping removes only the FCS field.
- FCS control for transmit/receive packets.
- Access to registers with support for burst processing.
- MAL for packet moving having one-cycle MAL slave latency.
- Independent, large (2KB for EMAC0 and EMAC1 in 460EX/EXr/GT, 16KB for EMAC2 and EMAC3 in 460GT only) transmit FIFOs and (16KB) receive FIFOs with programmable thresholds to minimize overruns and underruns and parity bits indication to the FIFOs and LPRA entries
- Multiple packet handling in transmit and receive FIFOs.
- Unicast, multicast, broadcast, and promiscuous address filtering capabilities.
- Two 256-bit hash filters for unicast and multicast packets.
- Automatic retransmission of collided packets.
- Rejection of runt packets before providing them to MAL.
- Programmable interface for connecting to a Gigabit PHY. A gigabit media independent interface (GMII) interface can be used for connection to the attached PHY layer device (used when 1000 Mbps mode is chosen).
- Programmable inter-packet gap to enable tuning for better system performance.
- Compliance with IEEE 802.3x standard packet-based flow control, including self-assembled control pause packet transmitting.
- Support for VLAN tag ID in compliance with IEEE Draft 802.3ac/D1.0 standard.
- VLAN tag insertion or replacement for transmit packets is a programmable option.
- Wake on LAN (WOL) handling.

- Programmable internal and external loop-back capabilities.
- Extensive error/status vector generation for each processed packet.
- Power management using a clock and power management (CPM) unit.

28.2 MAL and EMAC Communication

Figure 28-1 is a block diagram showing how the EMAC is connected to the MAL and the OPB. The MAL, as instructed by buffer descriptors, moves data over the 128-bit extended OPB (EOPB) to or from the EMAC. The 32-bit OPB provides the CPU with access to EMAC configuration, control and status registers by means of the PLB to OPB bridge.

Figure 28-1. EMAC in a Typical Ethernet Application



Note: Each EMAC is connected to the OPB for configuration and status register access. The EMACs are connected to the MAL through separate transmit and receive EOPB interfaces which operate independently of each other.

The packet transfer flow consists of three phases. These three phases are used to define the details of the EMAC-MAL protocol.

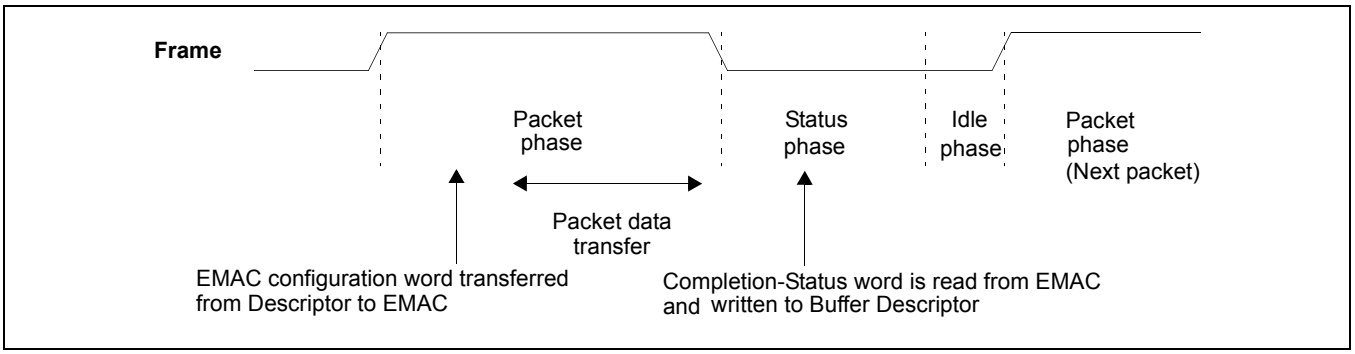
1. Packet phase - EMAC initiates a packet transfer operation. The packet transfer is started by a command write. During command write MAL provides control information for EMAC on a per-packet basis. Following the command write, MAL begins the data transfer, during which MAL transfers data between the buffers located in

the system's memory and EMAC. In transmit, the data is transferred from the system's memory to EMAC, while in receive, the data is transferred from EMAC to the system's memory buffers.

- EMAC remains in the packet phase until the data transfer has been completed or a ready status can be returned to MAL. The packet phase ends when EMAC deasserts the FRAME signal associated with the related channel (receive/transmit).
- The packet phase is defined by activity of an appropriate internal FRAME signal.

2. Status phase - This is the second phase of the packet transfer. Following the de-assertion of the FRAME signal, EMAC switches to the status phase. At this stage, EMAC uses an appropriate signal as a request for service which is interpreted by MAL as a request for status read.
3. Idle phase - EMAC moves into the idle phase following a reset or after status was transferred (end of status phase). During the idle phase, EMAC cannot send any signals to MAL, nor can MAL send any active signals to EMAC. EMAC exits the idle phase by asserting the FRAME signal (and entering the packet phase described above). Idle phase can be skipped when EMAC operates in multiple transfer mode.

Figure 28-2. EMAC-MAL Communication Phases

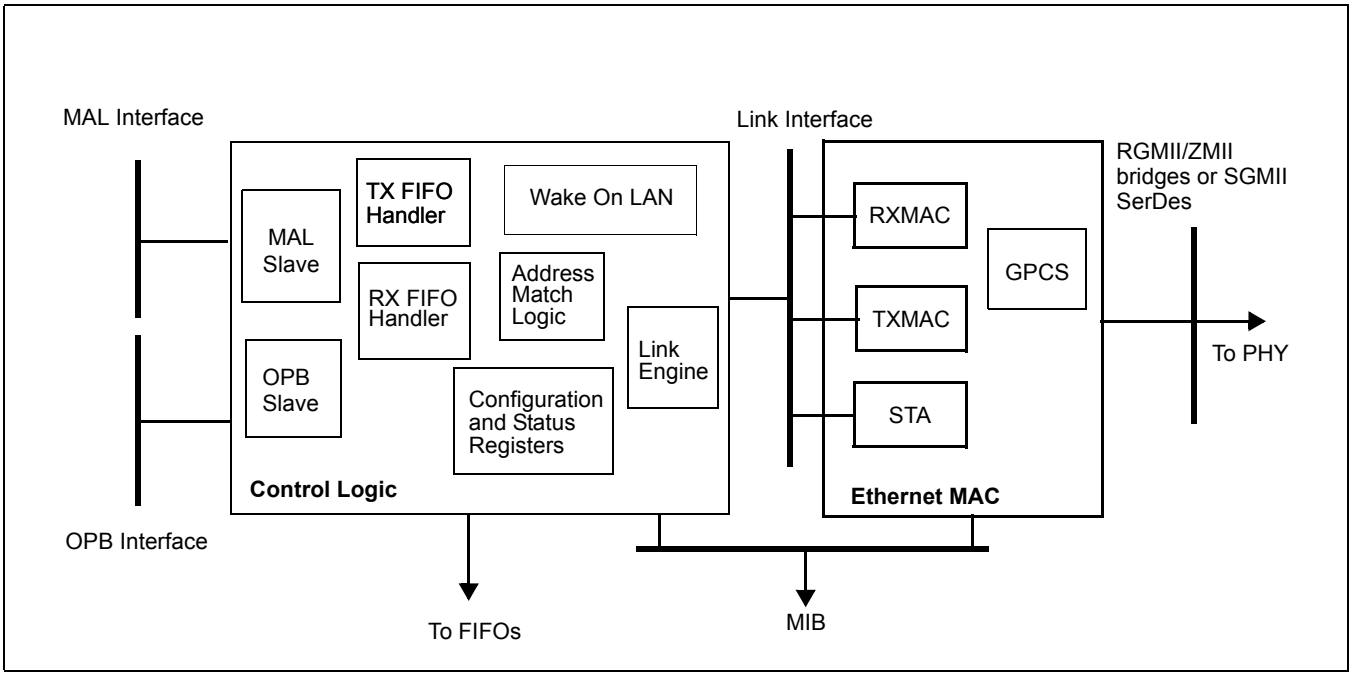


See *Figure 28-5* on page 1002 and *Figure 28-7* on page 1008 for more information about the EMAC-configuration word and Completion-Status word.

28.3 EMAC Operation

The EMAC hardware components and its internal structure are illustrated in *Figure 28-3*. The EMAC is connected to a physical layer device by means of the MII (for 10/100 Mbps Ethernet), the GMII (for 1000 Mbps Ethernet using an external GPCS), or the PMA interface (for 1000 MBps Ethernet using the EMAC internal GPCS). The active PHY interface (GMII or PMA) is defined in the EMAC configuration.

Figure 28-3. Internal EMAC Structure



The control logic sub-block implements the following functions:

- OPB slave device
- MAL slave device
- FIFO management logic
- Ethernet address and pause packet match logic (address match logic)
- Register file for FIFOs and Ethernet MAC handler management
- Logic for support of WOL technology
- Link engine

The Ethernet MAC sub-block implements the following functions:

- Transmit MAC Handler (TXMAC)
- Receive MAC Handler (RXMAC)
- MII/GMII management function unit (STA)
- Internal GPCS unit

The above functions are described in the following sections.

28.3.1 MAL Slave Logic

The MAL slave (MALS) logic controls MAL transactions. MALS transfers TX and RX data between MAL and the OPB on one side, and the EMAC FIFO handlers on the other side. MALS is a dedicated MAL slave.

28.3.2 OPB Slave Logic

The OPB slave (OPBS) logic controls all OPB transactions between the processor and the EMAC configuration and status registers.

28.3.3 FIFO Management Logic

The FIFO management logic is used for data interchange handling with the attached receive (RX FIFO), transmit (TX FIFO), and User Priority (QoS FIFO) modules.

28.3.4 Ethernet Address Match Logic

Address match logic checks the destination address of received packets against a set of predefined addresses specified by the current address filtering mode. EMAC contains one unicast (individual address) register, two hash tables for filtering individual and group address, and logic for detecting broadcast address (all ones). EMAC supports promiscuous mode and multicast promiscuous mode.

This logic also checks the destination address of the incoming packet against a special multicast address used for control (pause) packet recognition.

All checks for address matching are performed only after the entire destination address field is received (except for promiscuous and multicast promiscuous modes).

28.3.5 Configuration and Status Registers

Configuration and status registers define the EMAC configuration and reflect error/status of recent transmitted or received packets.

28.3.6 Wake On LAN Logic

EMAC supports Wake On LAN (WOL) technology, an industry standard described in the *Wired for Management (WFM)* specification. This technology allows a sleeping or powered-off network node to be awakened with a special packet called a Magic Packet. In the PPC460EX/EXr/GT, with WOL mode enabled, the EMAC discards all incoming packets and does not request data from the MAL for transmission. When a magic packet is detected, an interrupt is generated on UIC2 (bit 20 for EMAC0 or bit 21 for EMAC1).

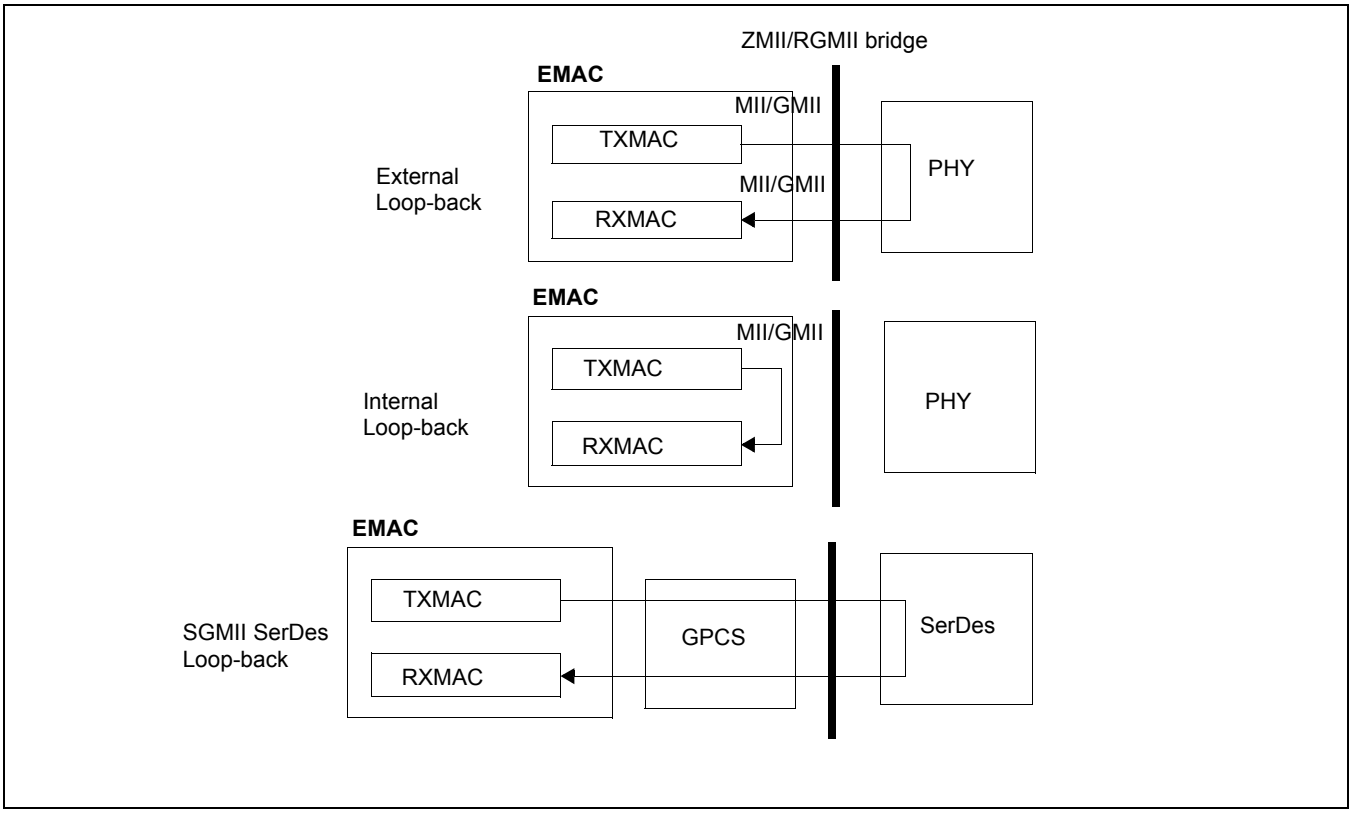
28.3.7 Ethernet MAC

The Ethernet MAC logic supports a media independent interface (MII), a gigabit MII (GMII) and a PMA interface.

28.3.8 EMAC Loopback Modes

EMAC supports the external and internal loop-back modes illustrated in the following figure.

Figure 28-4. EMAC Loopback Modes



External loop-back mode uses the PHY device. From the perspective of the EMAC, external loop-back is identical to full-duplex operation. Configuring EMAC to operate in a full-duplex mode enables external loop-back. EMAC does not need an external loopback configuration signal. In external loop-back with the internal GPCS, the GPCS loop-back register bit is set by means of the EMACx_STACR register.

In internal loopback mode, data from the EMAC transmit channel is routed to the EMAC receive channel. The loopback mode functions correctly with or without a connected PHY. However, without a PHY, the EMAC transmit and receive clocks must be provided internally by (enabling Ethernet clock select bit (SDR0_ETH_CFG[EMACx_PHY_CLK_SEL] = 1). To enable internal loop back, set EMACx_MR1[FDE] = 1 and EMACx_MR1[ILE] = 1.

For internal SGMII SerDes loopback, set SDR0_ETH_CFG[SGMIIx_LPBK] = 1 and GPCSx_CR[PL] = 1.

28.4 EMAC Transmit Operation

The TXMAC logic of EMAC performs transfer of data from the MAL device to the physical layer device (PHY). At the end of a frame's transmission, EMAC writes a completion status word into the frame buffer's descriptor. This allows software to check that a transmissions completed successfully or to determine which, if any, error occurred.

28.4.1 Arbitration Between TX Channels

EMAC's TX channel (TXMAC) can be configured to work in either of two ways: single packet mode or multiple packet mode.

In single packet mode, selected by setting EMACx_MR1[TR] = 0, the TXMAC requests a single Ethernet frame from MAL. Note that a Ethernet frame can consists of one or more data buffers. When TXMAC receives notification from the MAL that the transfer to the TX FIFO is done, TXMAC resets the Get Next Packet flag (EMACx_TMR0[GNP] = 0). The TXMAC does not request another transfer by the MAL until the device driver software again sets Get Next Packet (EMACx_TMR0[GNP] = 1).

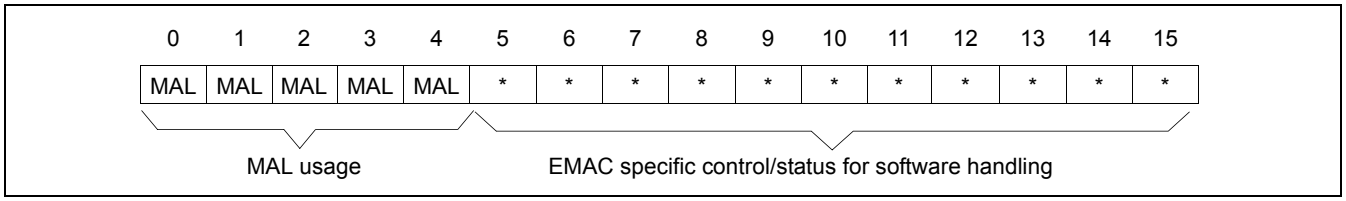
In multiple packet mode, selected by setting EMACx_MR1[TR] = 1, when the TXMAC receives indication from the MAL that the transfer to the TX FIFO is done, the channel asks MAL for the next packet (if there is enough room in the TX FIFO). The channel continues to request more packets until one of the following events occurs:

- The TXMAC receives notification from MAL that the next buffer descriptor is not marked ready for transmission. When this occurs, TXMAC sets EMACx_TMR0[GNP] = 0 and waits for software to reactivate the channel by setting EMACx_TMR0[GNP] = 1.
- A transmit error or signal quality error (SQE) occurs and the corresponding interrupt is not masked in the EMACx_ISER. After such an error, the channel sets EMACx_TMR0[GNP] = 0, and sets EMACx_ISR[DB] = 1 (the EMACx_ISR field that is set depends on which channel is active) and the corresponding EMACx_ISR error. The channel does not request service again until EMACx_TMR0[GNP] = 1 and EMACx_ISR[DB] = 0.

28.4.1.1 MAL TX Descriptor Control/Status Field

For each transmitted packet, MAL uses the descriptor control/status field of the buffer descriptor to provide an EMAC with buffer control information, and to store packet status read from the EMAC after transmission is complete (read). Software writes the control bits in the buffer descriptor before packet transmission is requested, and accesses the completion status from the buffer descriptor after packet transmission has completed. See *Buffer Descriptor (BD) Structure* on page 954 for more information on the buffer descriptor data structure.

Figure 28-5. MAL TX Descriptor Control/Status Field



Bits	Bit Name	Bit Description	Mode
0:4	MAL Usage	See *TX Buffer Descriptor MAL Control* on page 962	R
TX Control Information (Write Access by MAL to EMAC)			
5	Reserved	Always 0	W
6	Generate FCS	0 FCS is not generated by EMAC. 1 EMAC calculates and adds the FCS field to the packet to be transmitted.	W
7	Generate padding	0 Padding is not generated by EMAC. 1 EMAC adds the padding field to the packet to be transmitted (only when Generate FCS is also set).	W

Bits	Bit Name	Bit Description	Mode
8	Insert source address	0 EMAC will not insert source address. 1 EMAC inserts the source address field into the packet to be transmitted using the content of the Individual Address High (EMACx_IAHR) and Individual Address Low (EMACx_IALR) Registers.	W
9	Replace source address	0 EMAC will not replace source address. 1 EMAC replaces the source address field in the packet to be transmitted using the content of the Individual Address High (EMACx_IAHR) and Individual Address Low (EMACx_IALR) Registers.	W
10	Insert VLAN Tag	0 EMAC will not insert a VLAN tag. 1 EMAC inserts the VLAN Tag field into the packet to be transmitted using the content of the VLAN TPID register (EMACx_VTPID).	W
11	Replace VLAN Tag	0 EMAC will not replace the VLAN tag. 1 EMAC replaces the VLAN Tag field in the packet to be transmitted using the content of the VLAN TPID register (EMACx_VTPID).	W
TX Status Information (Read Access by MAL from EMAC)			
5	Parity Error	Indicates that the EMAC detected a parity error in the TX FIFO. This bit is interpreted by the MAL as an End of Buffer interrupt. The MAL stops writing the buffer to the EMAC. The parity error is recorded by EMACx_ISR[TxPE].	R
6	Bad FCS on transmitted frame	0 FCS was correct in the transmitted packet. 1 Indicates that a bad FCS was found in the transmitted packet (this bit is also set when parity error bit 5 is set).	R
7	Reserved	Always zero. (However this bit is used by TCP/IP hardware to report TX error.	R
8	Loss of carrier sense	0 No loss of carrier. 1 During the transmission of a frame, the GMCCrS input was de-asserted after it previously was asserted, or it was not asserted at all. Applicable only in half-duplex mode.	R
9	Excessive deferral	0 No excessive deferral. 1 Indicates that the current frame has been deferred for an excessive period of time. Applicable only in half duplex mode. The value of this period in bit times is calculated in the following ways: For 10/100 Mbps operation it is: 2 x (maxFrameSize x 8) bit times. For 1000 Mbps operation it is: 2 x (burstlimit+maxFrameSize x 8+headerSize) bit times. The burstlimit is configured by SDR0_MALTBL.	R
10	Excessive collisions	0 Less than 16 collisions. 1 Indicates that the current frame transmission had ended with a collision on the 16th consecutive attempt. Applicable only in half-duplex mode.	R
11	Late collision	0 No late collision. 1 Frame collided outside of the collision window. Applicable only in half-duplex mode.	R
12	Multiple collision	0 More than 1 but less than 16 collisions did not occur. 1 Transmitted frame collided more than once but less than 16 times. Applicable only in half-duplex mode.	R

Bits	Bit Name	Bit Description	Mode
13	Single collision	0 Single collision did not occur. 1 Activates if transmitted frame collided once. Applicable only in half-duplex mode.	R
14	Underrun	0 Underrun did not occur. 1 Frame transmission was aborted because of underrun; data from the Transmit FIFO was not valid in time to allow continuous data transmission on the GMII interface.	R
15	SQE	0 Signal Quality Error did not occur. 1 Signal Quality Error test failed during packet transmission. Applicable only in half -duplex mode during 10 Mbps operation.	R

28.4.1.2 Early Packet Termination during Transmit

EMAC can initiate early packet termination during transmit, terminating packet transmission before MAL finishes transferring all packet data from memory to the EMAC transmit FIFO. Early packet termination is typically used when error conditions force the EMAC to abort a transmission.

EMAC performs early termination on the MAL interface if any of the following conditions occur:

- Underrun in the transmit FIFO.
- Excessive collisions.
- Excessive deferral.
- Late collision.

28.4.1.3 Empty Packets

EMAC treats empty packets as if a normal packet has been cleared, but does not insert any data to the transmit FIFO. A completion-status word of all 0s is returned to the buffer descriptor after an empty packet is processed. EMAC expects that for quadword-aligned (16 bytes) packets, MAL requests the completion-status word during the last packet-data transfer, rather than providing an empty packet indication.

28.4.1.4 Automatic Retransmission of Colliding Packets

EMAC automatically retransmits packets that collide on the MII/GMII interface. The transmit FIFO always preserves the first 64 (10/100 Mbps) or 512 (Gigabit Ethernet media) bytes of a packet until it receives an indication that the collision window has passed. Otherwise, if a collision was detected within the collision window, the packet is retransmitted without a new request from MAL.

28.4.1.5 Inter-Packet Gap (IPG) Tuning

The inter-packet gap (IPG), is adjustable using the EMACx_IPGVR. Changing the contents of EMACx_IPGVR enables the user to adjust the fairness or aggressiveness of link utilization by the EMAC on the physical media. Programming a lower value (but not less than 96 bit times, the minimum allowed) causes EMAC to not wait for others to acquire a half-duplex link (it more aggressively utilizes the link). This can result in EMAC capturing the network by forcing less aggressive nodes to find the link busy. Programming a larger number of bit times causes EMAC to wait longer before checking if the link is busy, therefore, it might defer more often to other nodes. EMAC throughput performance decreases as the IPG period is increased from the minimum value, but the resulting behavior can improve media performance by reducing the occurrence of collisions.

28.4.1.6 Full-Duplex Operation

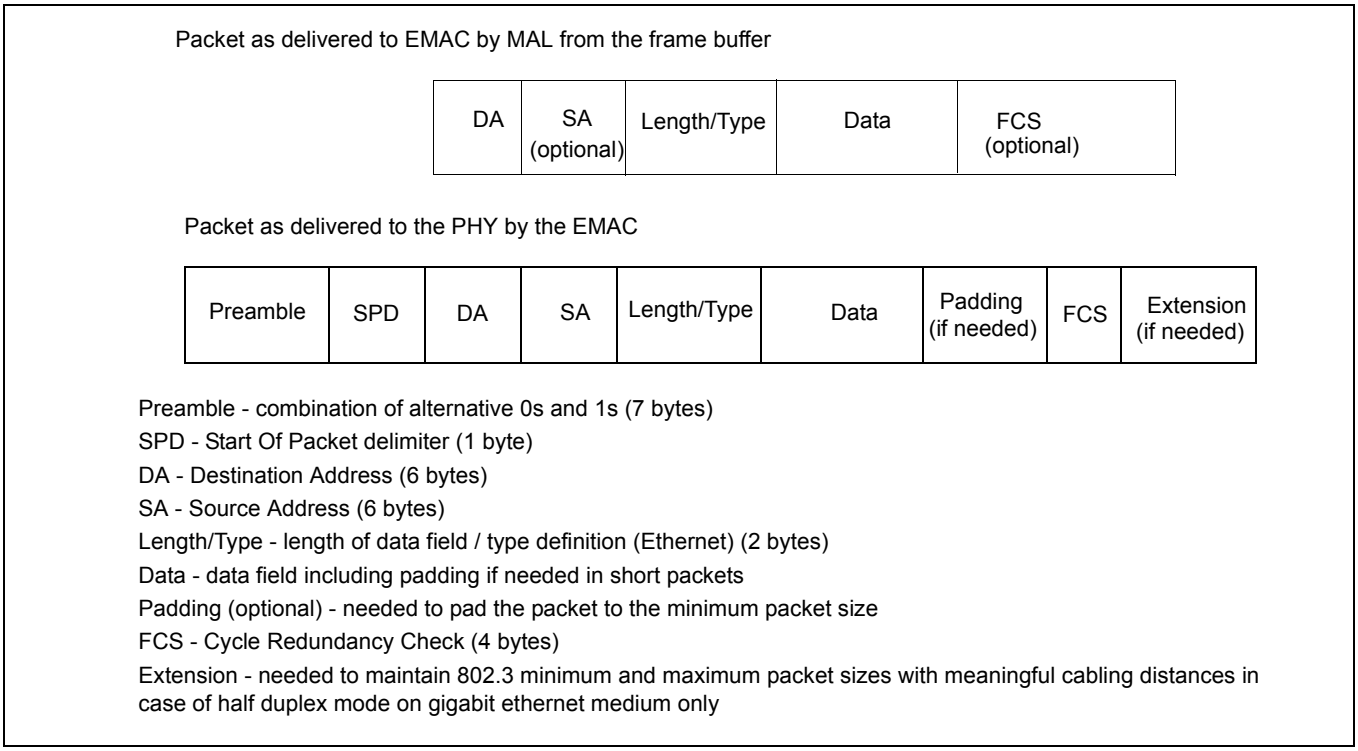
Full-duplex operation allows simultaneous transmit and receive activity on the GMII interface. Software can set EMACx_MR1[FDE] = 1 to enable full-duplex mode. During full-duplex operation, the following changes occur in EMAC functionality.

- Transmission is not deferred while receive is active.
- The IPG counter, which controls transmit deferral during the IPG between back-to-back transmits, is started when transmit activity for the first packet ends, instead of when transmit and carrier activity end.
- SQE test is not performed.
- Collision indication is ignored.

28.4.1.7 Packet Content Configuration Options

EMAC can modify the content of the packet coming from MAL before issuing it to the MII/GMII interface. *Figure 28-6* illustrates possible changes in the transmit packet format.

Figure 28-6. Transmit Packet Structure (Excluding VLAN Tagged and Control Packets)



The following options, unless mutually exclusive, can be set for each packet, and can be provided as a part of command write transactions from MAL (see *MAL TX Descriptor Control/Status Field* on page 1002).

- Automatic data padding for short transmit packets

  EMAC pads the transmit packet with extra bytes between the data and the FCS field to reach a total length of 64 bytes (including FCS). This feature is supported only when the packet coming from the transmit FIFO does not contain the FCS. Automatic padding enables software to avoid sending padding as a part of the packet data field, and, therefore, reduces the amount of data transferred on the system bus during the short packet transmission.

- Source address insertion

  EMAC adds the source address (SA) field to the transmitted packet. EMAC uses the contents of the Individual Address High (EMACx_IAHR) and Individual Address Low (EMACx_IALR) registers for the source address value. *This option is mutually exclusive with source address replacement.*

- Source address replacement

  EMAC replaces the source address received from the transmit FIFO with the contents of EMACx_LSAH and EMACx_LSAL. *This option is mutually exclusive with source address insertion*.

- Add four FCS bytes

  EMAC calculates the FCS for the transmitted packet. The FCS is appended to data coming from the Transmit FIFO or padding bytes (if added).

- VLAN Tag insertion

  EMAC adds the content of the VLAN Tag field to the transmitted packet (see *VLAN Support* on page 1014). EMAC uses the content of the VLAN TPID (EMACx_VTPID) and VLAN TCI (EMACx_VTCI) registers for the VLAN Tag value. This feature is supported only when the packet from the transmit FIFO does not contain an FCS. *This option is mutually exclusive with VLAN tag replacement*.

- VLAN Tag replacement.

  EMAC replaces the content of the VLAN Tag field in the transmitted packet. EMAC uses the contents of EMACx_VTPID and EMACx_VTCI for the VLAN Tag value. This feature is supported only when the packet from the transmit FIFO does not contain an FCS. *This option is mutually exclusive with VLAN tag insertion*.

Table 28-1 summarizes the possible options for adding FCS and SA to the transmitted packet.

Table 28-1. FCS/SA Enable - Possible Configurations

Configuration Options			EMAC Action		
Generate FCS	**Insert SA**	**Replace SA**	**Add FCS**	**Add SA**	**Replace SA**
0	Don't care	Don't care	N	N	N
1	0	0	Y	N	N
1	0	1	Y	N	Y
1	1	0	Y	Y	N

Table 28-2 summarizes the possible options for adding FCS and padding to the transmitted packet.

Table 28-2. FCS/Pad Enable - Possible Configurations

Configuration Options		EMAC Action	
Generate FCS	**Generate Pad**	**Add FCS**	**Add Pad**
0	Don't care	N	N
1	0	Y	N
1	1	Y	Y

Table 28-3 summarizes the possible options for adding FCS and VLAN Tag to the transmitted packet.

Table 28-3. FCS/VLAN Tag Enable - Possible Configurations

Configuration Options			EMAC Action		
Generate FCS	**Insert VLAN Tag**	**Replace VLAN Tag**	**Add FCS**	**Insert VLAN Tag**	**Replace VLAN Tag**
0	Don't care	Don't care	N	N	N
1	0	0	Y	N	N
1	0	1	Y	N	Y
1	1	0	Y	Y	N

28.5 BEB Back Off Algorithm

The Backoff algorithm is a randomizing algorithm to determine the retransmission time of the collided Tx frame. The standard algorithm is the Binary Exponential Backoff (BEB). According to BEB following a collision, TXMAC must wait one back off period before trying to retransmit the frame. The back off period is an integer number of slot times chosen as a uniformly distributed random integer r in the range:

$0?r<2^k$, where k = min(n,10) and n is a number of collisions.

After the back off period elapses, TXMAC, if there is pending frame, starts transmission as soon as the IFG condition is satisfied.

28.6 FCS Calculation

A CRC (Cyclic Redundancy Check) is used by the TXMAC to generate a 32-bit CRC value for the FCS (Frame Check Sequence) field. This value is computed as a function of the transmitted packet's complete contents, excluding preamble, SPD, and extension fields.

The following polynomial is used to generate the CRC (according to IEEE Standard 802.3z) needed for transmit FCS and for checking received packet FCS:

$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

In addition to checking for FCS errors, the EMAC's receive logic (RXMAC) can use the eight msbs of the calculated FCS for a hash table Ethernet address filter. See *Ethernet Address Filter Operation* on page 1019.

28.7 EMAC Receive Operation

The receive part of EMAC is responsible for receiving packets coming from the physical layer (PHY) device (by means of the MII/GMII) and forwarding them to the receive channel of the attached MAL. At the end of the reception process, EMAC provides a status/error word that enables software to monitor the receive operation.

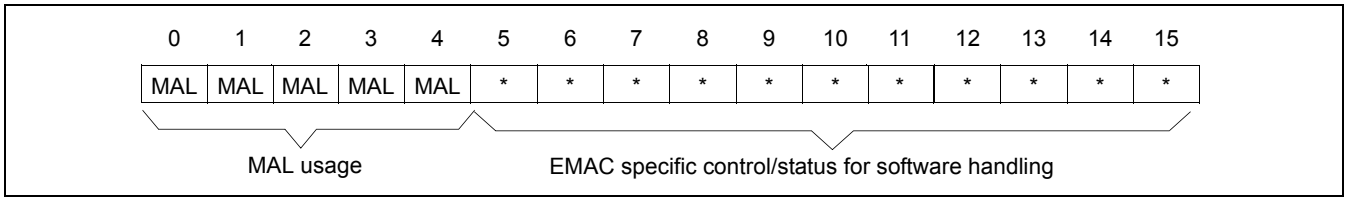
28.7.1 EMAC – MAL RX Packet Transfer Flow

EMAC initiates request for service from MAL when the number of occupied entries in the receive FIFO reaches the low water mark specified in EMACx_RWMR[RLWM].

28.7.2 MAL RX Descriptor Status

For each packet that is received, MAL obtains status from EMAC after reception is complete, and writes this information into the buffer descriptor status/control field. Software uses this information to monitor the status of received packets.

Figure 28-7. MAL RX Descriptor Control/Status Field



Bits	Bit Name	Bit Description	Mode
0:4	MAL Usage	See *RX Buffer Descriptor MAL Control Options* on page 963.	R
RX Status Information (Read Access by MAL from EMAC)			
5	Parity Error	Indicates that the EMAC detected a parity error in the RX FIFO. This bit is interpreted by the MAL as an End of Buffer interrupt. The MAL stops reading the packet from the EMAC. The parity error is recorded by EMACx_ISR[RxPE].	R
6	Overrun Error	0 No overrun error. 1 EMAC detected an overrun error. An overrun error occurs if the flow of received data to the RX FIFO is corrupted because of insufficient empty space.	R
7	Pause Packet	0 Received packet is not a control pause packet. 1 Received packet is a control pause packet.	R
8	Bad Packet	0 No packet errors. 1 Early termination caused by packet error.	R
9	Runt Packet	0 Duration of GMCRxDv signal OK. 1 Duration of GMCRxDv signal greater than ShortEventMax Time constant and less than collision windows.	R
10	Short Event	0 Duration of GMCRxDv signal OK. 1 Duration of GMCRxDv signal was less than ShortEventMaxTime constant	R
11	Alignment Error	0 Received packet length OK. 1 Received packet length not an integral number of octets.	R
12	Bad FCS	0 FCS OK. 1 The FCS value does not match the FCS value calculated by EMAC.	R

Bits	Bit Name	Bit Description	Mode
13	Packet Too Long	0 Received packet length OK. 1 Received packet length exceeds maximum packet length. 1518 bytes for standard packet (checked only when the Length/Type field of the transmitted packet contains a Length value, and Jumbo packet size support is disabled) 1522 bytes for VLAN tagged packet (checked only when the Length/Type field of the transmitted packet contains a Length value, and Jumbo packet size support is disabled) 9018 bytes when Jumbo packet size support is enabled, with no VLAN tag support 9022 bytes when Jumbo packet size support is enabled, with VLAN tag **Note:** The data following the maximum packet length is not transferred to MAL	R
14	Out of Range Error	0 Received packet Length/Type field value is in the allowed range (≤1500 bytes). 1 Received packet Length/Type field value is greater than the 1500 bytes maximum allowed by IEEE 802.3 or DIX specification for Logical Link Control protocol (LLC) payload data size. The received data sizes which cause this error are in the range 1501 bytes to 1535 bytes. Actual received data sizes greater than 1535 bytes (when Jumbo sized packet support is disabled) cause a Packet Too Long error.	R
15	In Range Error	0 Received packets Length/Type field value is 1500 bytes or less, and matches the actual received length as counted by the RXMAC logic. 1 Received packets Length/Type field value is 1500 bytes or less, and is different then the actual received length as counted by the RXMAC logic. Refer to *Table 28-4* for a description of conditions for activating this status bit.	R

Table 28-4. In Range Length Error Behavior for Various Packet Lengths

Programmed Length (Bytes)	Actual Length	EMAC Action
Less than 46 (42 if VLAN Tagged packet)	Differs from 46 (42 in case of VLAN Tagged packet)	In range length error is activated
Less than 46 (42 if VLAN Tagged packet)	46 (42 in case of VLAN Tagged packet)	In range length error is not activated
Greater than or equal to 46 (42 if VLAN Tagged packet) and less than or equal to 1500	Equals the length field value	In range length error is not activated
Greater than or equal to 46 (42 if VLAN Tagged packet) and less than or equal to 1500	Differs from the length field value	In range length error is activated

28.7.3 Early Packet Termination during Receive

Early packet termination occurs when packet reception is aborted by EMAC before the packet data transfer to MAL is completed.

EMAC performs early termination in the following cases.

- An overrun occurs in the receive FIFO
- Packet is too long and the Receive Oversize Packet option is not enabled (EMACx_RMR[ROP] = 0)

28.7.4 Discarding Packets During Receive

Received packets can be discarded if certain error conditions are detected. EMAC behavior depends on whether the packet to be discarded is already being output to MAL. If the packet containing the error is not being provided to MAL when the discard condition is detected, the packet is flushed from the Receive FIFO. In this case, EMAC does not provide status information to MAL. If the packet containing the error is already being output to MAL, EMAC initiates an early packet termination procedure, as described in *Early Packet Termination during Receive* on page 1009

Each receive discard condition can be individually controlled using appropriate settings, as described in *Receive Mode Register (EMACx_RMR)* on page 1026.

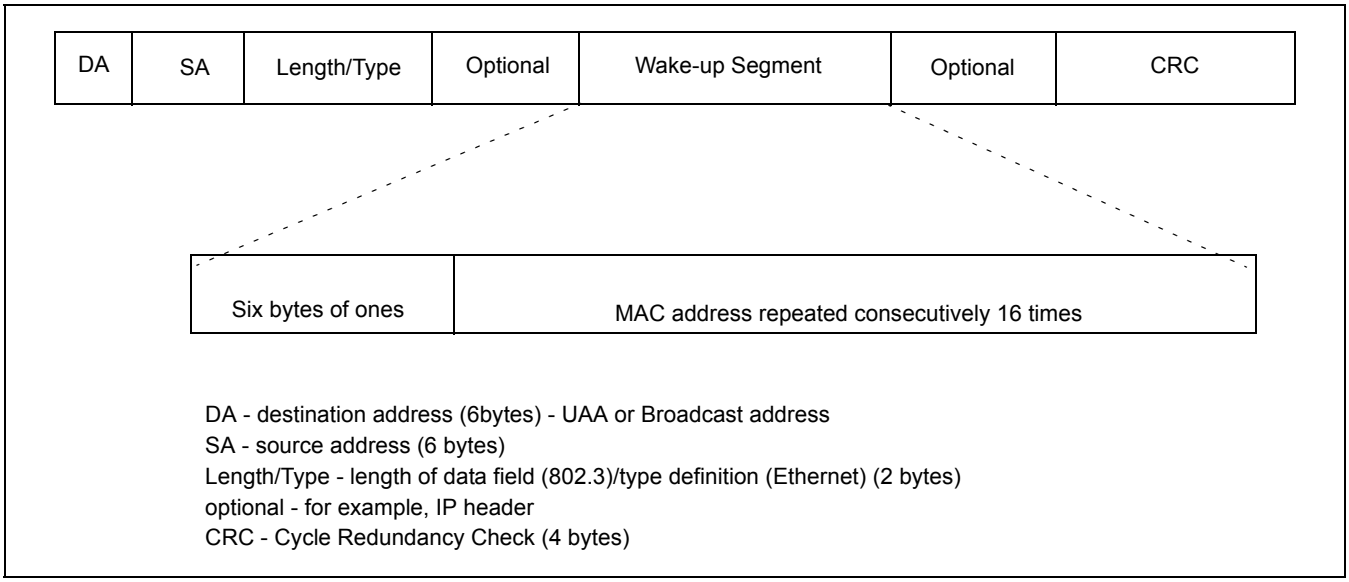
28.7.5 Wake Up On Lan (WOL) Support

WOL logic in EMAC supports WOL technology, an industry standard in the Wired for Management (WFM) Specification. WOL remotely awakens sleeping or powered-off nodes on a network. To wake up a node, a specific packet, called a *magic packet*, is sent to the network node. When so configured, EMAC monitors the incoming bit stream for magic packets.

The magic packet, also called a wake-up packet, contains a unique data field not normally expected in LAN traffic. When a WOL-enabled adapter on a powered-off client decodes this data field, a wake-up signal is generated. In the PPC460EX/EXr/GT, with WOL mode enabled, EMAC discards all incoming packets and does not request data from the MAL for transmission. When a magic packet is detected, an interrupt is generated on UIC2 (bit 20 for EMAC0 or bit 21 for EMAC1).

Figure 28-8 shows the wake-up packet format. The key to the wake-up packet is the MAC address of the target client, which is repeated 16 times. This pattern of 16 addresses in the data field is not expected to occur in any packet except the wake-up packet.

Figure 28-8. Wake-Up Packet Format



The destination address can be a specific address, called the universally administered address (UAA), or a broadcast address. If the destination address is a UAA, the wake-up packet is sent only to the client at that address. However, because the client is powered off and is no longer transmitting, some protocols remove the client MAC address from routing tables and internal caches at other nodes. In this case, wake-up packets

addressed to a target client are discarded because nodes and routers do not know where to send them. The solution to this problem is to use a broadcast address. A directed broadcast has a valid network address and a broadcast host address. Network routers and nodes forward directed broadcasts to the appropriate network, where it is seen as a MAC-level broadcast and detected by the powered-off client.

28.7.5.1 EMAC WOL Support

EMAC enters WOL mode when EMACx_MR0[WKE] = 1.

WOL mode should only be changed while EMACx_MR0[RXI] = 1 and EMACx_MR0[RXE] = 0. After EMACx_MR0[WKE] = 1, EMACx_MR0[RXE] can be set to 1.

A reset (soft or hard) should be issued before programming EMAC to WOL mode. In WOL mode, EMAC does not propagate any received packets to MAL. Also, EMAC transmit channels do not request data from MAL.

28.8 Flow Control

For efficient system performance, each EMAC implements full-duplex flow control by handling specific MAC control packets contained in the pause opcode. EMAC supports flow control as defined in the IEEE 802.3x-1997 standard.

28.8.1 MAC Control Packet

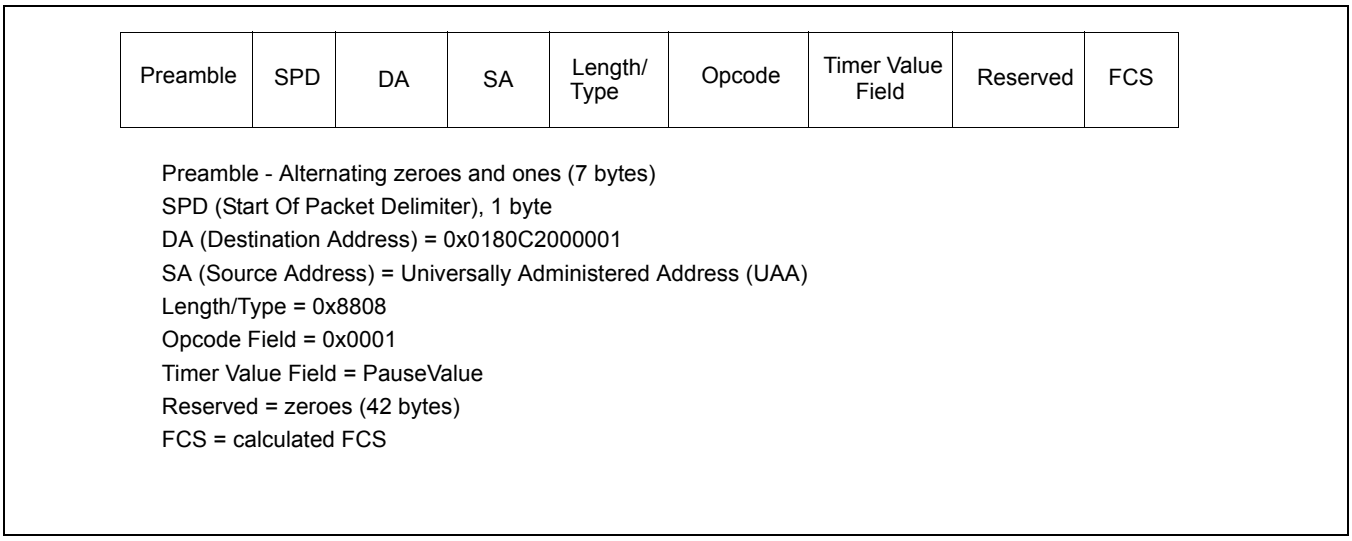
The flow control mechanism enables receive FIFO control logic to automatically notify the node transmitting packets to suspend transmission for a defined period of time. The pause control packet has a fixed length defined as follows:

MinFrameSize – 32 bits (60 bytes)

MAC control packets have a unique value of 0x8808 in the length/type field, and share the same packet format as normal Ethernet packets, except that the data field consists of an opcode field and a parameter field. The opcode field contains an opcode command and the parameter field contains a value associated with the opcode command (0x0001). The only opcode command defined by IEEE 802.3x is the pause opcode; the parameter field for the pause opcode defines the pause time. MAC control packets containing a pause opcode, also called pause packets, should have a destination address equal to a reserved multicast address. The reserved multicast address is 0x0180C2000001.

Note: This reserved multicast address should also be mapped into one of 64 categories corresponding to 64 bits stored in the EMACx_GAHT1:4 registers.

Figure 28-9. Control Packet Format



The timer value field contains the value of the delay interval in resolution of pause_quanta, defined in IEEE 802.3x as follows: "MAC Control Parameter[s] (pause_time) is a 2-octet, unsigned integer containing the length of time for which the receiving station is requested to inhibit data packet transmission. The field is transmitted most-significant octet first, and least-significant octet second. The pause_time is measured in units of pause_quanta, equal to 512 bit times. The range of possible pause_time is 0 to 65535 pause_quanta."

28.8.2 Control Packet Transmission

Three options initiate the pause packet transmission from EMAC. The transmitted pause packet forces the node, with the destination address specified, to temporarily suspend the transmission of packets to EMAC.

1. Software initiated

   The packet transferred to EMAC by MAL for transmission is a pause packet created by software. EMAC transmits this as a normal packet.

2. Automatic flow control initiated

   The EMAC integrated flow control mechanism detects the need for and then transmits a control (pause) packet automatically. When building the control packet, EMAC obtains the SA (source address) field from EMACx_IAHR and EMACx_IALR, and the timer value from the Pause Timer Register (EMACx_PTR[TVF]). The contents of the other fields in the packet are shown in *Figure 28-9*.

3. Software initiated pause packet activated by setting the issue pause bit, EMACx_TPC[IPA] = 1.

   Once set the EMAC generates and transmits a self-assembled control (pause) packet, the value of the Source Address (SA) field is take from the content of EMACx_IAHR and EMACx_IALR. The timer value is taken from the content of EMACx_TPC[TV].

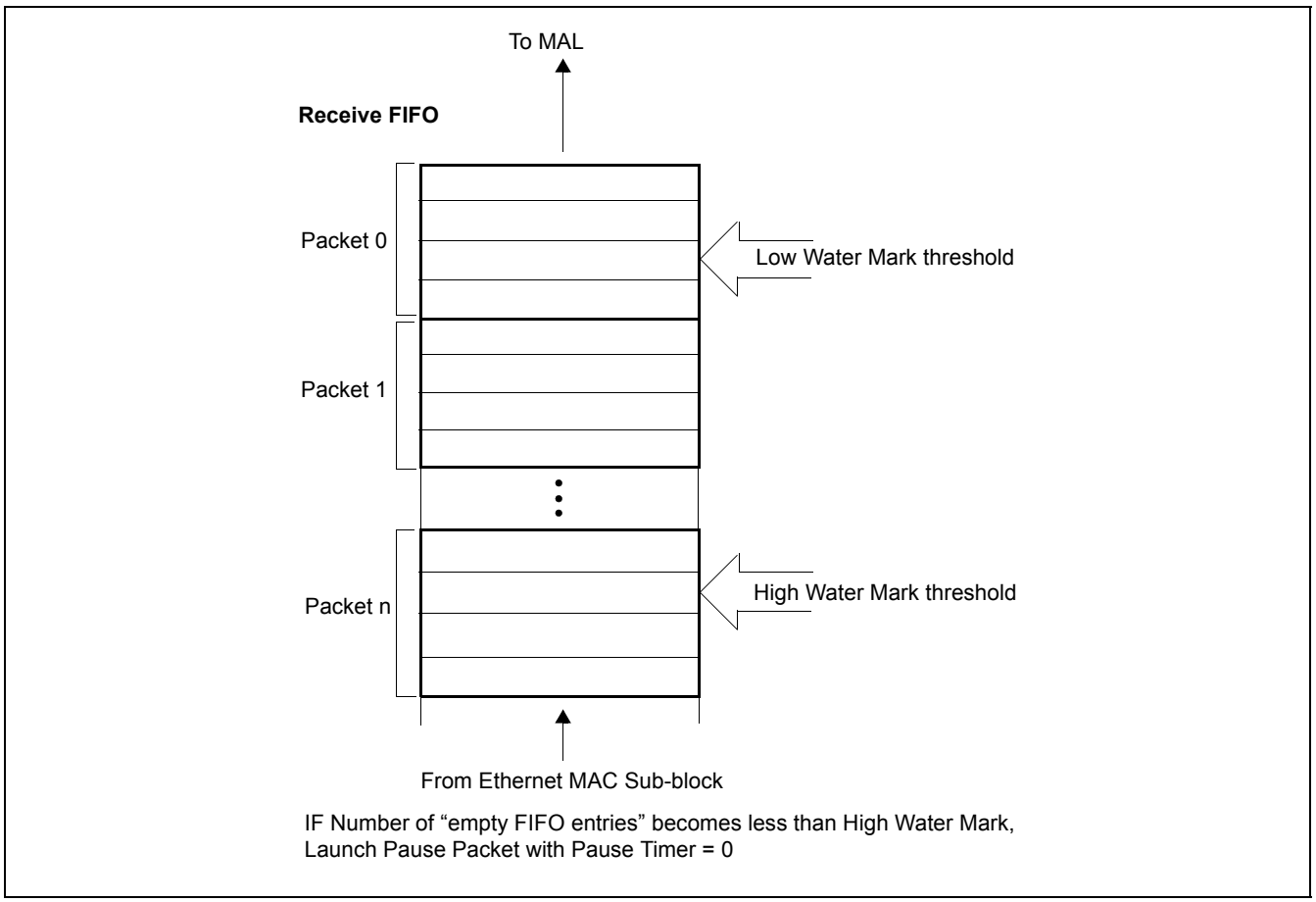
28.8.3 Integrated Flow Control

To enable integrated flow control in full-duplex mode, set EMACx_MR1[EIFC] = 1. EMAC sends a control packet of type Pause, with a programmable pause timer value, when the number of vacant FIFO entries in the Receive FIFO becomes less than the Receive High Water Mark (EMACx_RWMR[RHWM]), and a new packet enters the receive FIFO.

The Receive High Water Mark also specifies the number of FIFO entries to be occupied before a urgent (high priority) request for MAL transfer service is made.

Note: A 'FIFO entry' refers to the width of a FIFO array row which is 16 bytes (128 bits).

Figure 28-10. Integrated Flow Control Mechanism

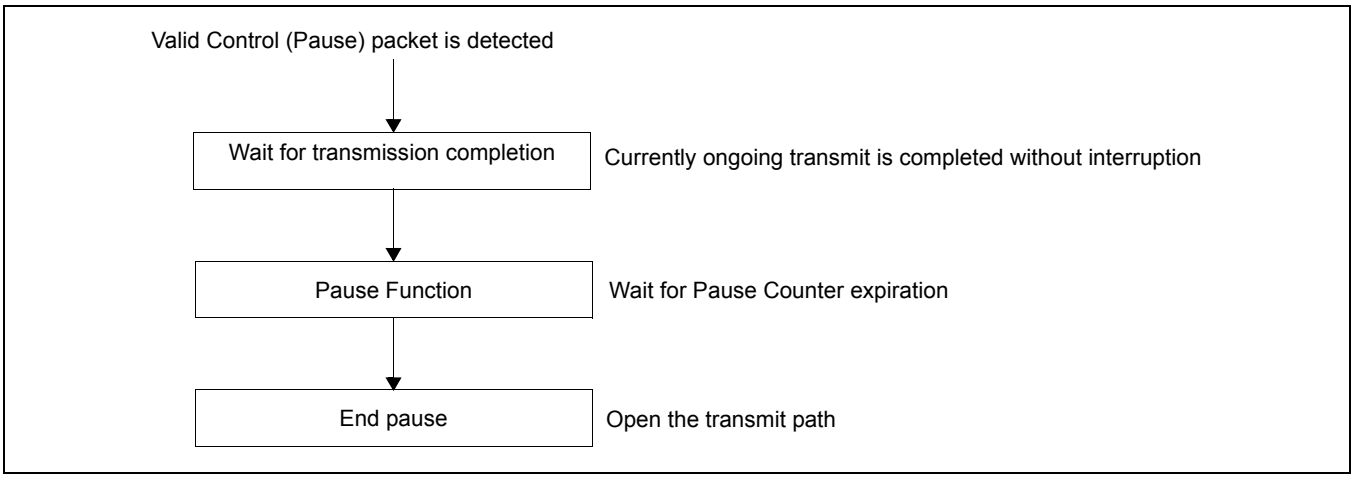


28.8.4 Control Packet Reception

In the receive path, the EMAC can be configured to respond to pause packets, or ignore them, as specified by EMACx_MR1[APP]. In order for the EMAC to process incoming Pause Packets, the EMACx_RMR [MAE] and EMACx_RMR[BAE] bits must be set to 1. When response to pause packets is enabled and EMAC detects a valid MAC control packet with a pause opcode, the EMAC stores the value of the timer value field. The received packet is considered a valid control packet only if no error was detected during the packet reception. If, at the end of packet reception, the packet is considered valid, EMAC launches a pause operation state machine, as specified in the IEEE P802.3x standard. *Figure 28-11* illustrates the pause operation state machine.

If a control (pause) packet is received while another packet is transmitted, the ongoing transmission process is completed and the transmitter is paused. If other packets are in the transmit FIFO, their transmission is delayed until the pause timer expires. EMAC normally does not pass the MAC control packets to MAL unless EMACx_RMR[PPP] = 1.

Figure 28-11. Pause Operation State Machine



In the Pause Function state, EMAC decrements its internal pause timer, which was set to the timer value field of the received control packet.

Note 1: The transmission of control (pause) packets is not affected by the reception of a receive control (pause) packet. Received control (pause) packets inhibit only the transmission of regular packets from the Transmit FIFO.

Note 2: Receipt of a new valid control (pause) packet causes the pause timer of EMAC to be reloaded with the contents of the timer value field of the recently received packet, regardless of the current pause timer setting. This indicates new pause operations take precedence over earlier pause operations.

28.9 VLAN Support

EMAC can handle VLAN tagged packets, as specified in IEEE Draft P802.3ac/D1.0a standard when EMACx_MR1[VLE] = 1.

A VLAN tagged frame is an extension of the standard MAC packet. The extension for VLAN tag support consists of a 4-octet VLAN tag inserted between the end of the source address and the beginning of the length/type fields of the MAC packet.

The VLAN tag consists of two fields:

- A 2-octet constant Type field value equal to the VLAN Tag Protocol Identifier (0x8100)
- A 2-octet field containing Tag Control Information (TCI)

The MAC client data and FCS fields of the basic MAC packet follow the VLAN tag. The length of the packet is extended by four octets by the VLAN tag (up to 1522 bytes and 9022 bytes for standard and Jumbo frame respectively). The FCS is calculated over all fields from the destination address through the end of the MAC client data or pad (if present); that is, all fields except the preamble, SPD, and FCS.

Figure 28-12. VLAN Tagged Packet Format

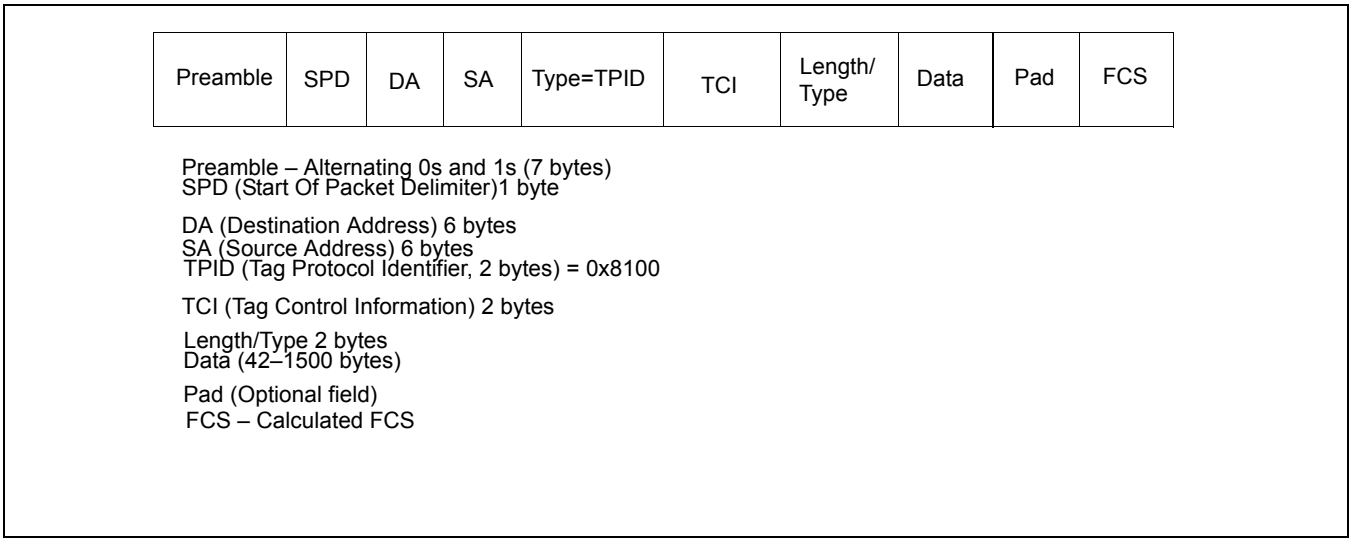
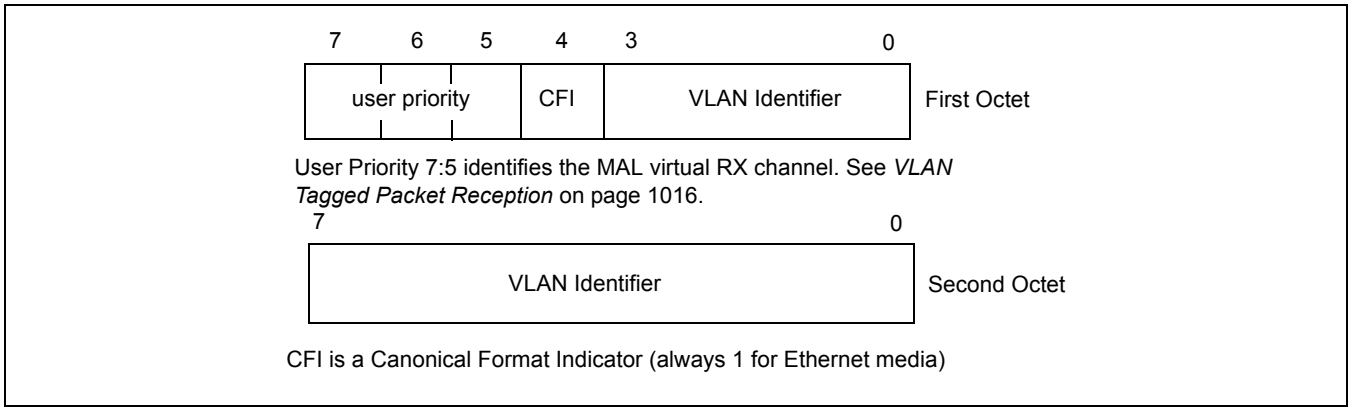


Figure 28-13. Tag Control Information Field Structure



28.9.1 VLAN Tagged Packet Transmission

When EMACx_MR1[VLE] = 1, the following configuration options are available, depending on the content of the appropriate bits in the MAL control word (See *MAL TX Descriptor Control/Status Field* on page 1002.)

- The Generate FCS bit 6 is not set or both Insert VLAN Tag and Replace VLAN Tag bits 10 and 11 are not set: EMAC transmits the packet without any changes
- Bit 6 is set and bit 10 is also set: EMAC will insert TPID and TCI for the transmitting packet using the content of EMACx_VTPID and EMACx_VTCI, respectively
- Bit 6 is set and bit 11 is also set: EMAC will replace TPID and TCI for the transmitting packet using the content of EMACx_VTPID and EMACx_VTCI, respectively

28.9.2 VLAN Tagged Packet Reception

When EMACx_MR1[VLE] = 1, the EMAC parses the VLAN Tag unique type/length in the incoming packet during the receive process. If the VLAN Tag is equal to the value stored in EMACx_VTPID, EMAC continues the receive process and allows the received packet to contain up to 1522 octets and 9022 when Jumbo packets support is enabled. Otherwise, the receive process is continued unless the length is greater than 1518 bytes 9018 when Jumbo packets support is enabled.

When delivering the VLAN tagged frame to the MAL, the EMAC provides QoS information with the frame in the form of side band signals, QoS0:2. QoS0:2 assign priority to a frame based on the User Priority bits 7:5 of the VLAN tag as shown in *Figure 28-13*. The priority assignment identifies the MAL Virtual RX Channel. There are eight virtual RX channels per EMAC.

QoS0:2/User Priority 7:5	MAL Virtual RX Channel
000	Virtual RX Channel (N*8 + 0)
001	Virtual RX Channel (N*8 + 1)
010	Virtual RX Channel (N*8 + 2)
011	Virtual RX Channel (N*8 + 3)
100	Virtual RX Channel (N*8 + 4)
101	Virtual RX Channel (N*8 + 5)
110	Virtual RX Channel (N*8 + 6)
111	Virtual RX Channel (N*8 + 7)

N is the EMAC number. For EMAC0, N is 0 and EMAC1, N is 1, etc.

28.9.3 Address Match Mechanism

The address match (or filtering) mechanism is a hardware aid that reduces the average amount of CPU cycles required to determine whether an incoming packet should be accepted.

EMAC uses various address filters for incoming packets by using the following address recognition modes.

- Individual mode (also referred to as physical)
- Multicast mode (also referred to as group)
- Broadcast mode (an all-ones group address)
- Promiscuous mode
- Promiscuous multicast mode
- WOL mode

A flowchart for address recognition of received packets is shown in *Figure 28-14* on page 1018. If the least significant bit (lsb) of the first byte of the destination address (DA) is 0, the packet is considered individual. If the first bit received is 1, the packet is considered multicast. When the DA field contains all 1s, the packet is broadcast, a special type of multicast.

28.9.3.1 Non-WOL Mode

When EMAC operates in single individual mode (EMACx_RMR[IAE] = 1), the DA of the received packet is compared to the physical address stored in EMACx_IAHR and EMACx_IALR.

When EMAC operates in multiple individual mode (EMACx_RMR[MIAE] = 1), EMAC performs a calculation on the contents of the DA field (logical address filter) to determine whether or not to accept the packet.

When EMAC operates in promiscuous mode (EMACx_RMR[PME] = 1), all properly formed packets are received, regardless of the content of the DA field.

When EMAC operates in multicast promiscuous mode (EMACx_RMR[PMME] = 1), all multicast packets are received, regardless of the content of the DA field.

When EMAC operates in broadcast address mode (EMACx_RMR[BAE] = 1), EMAC performs an address compare on received packets with broadcast addresses.

When the EMAC operates in non-IP multicast address mode (EMACx_RMR[MAE] = 1 and EMACx_RMR[NIPMAE]=1) , EMAC performs the following operations:

1. Applies a bit wise AND function between the multicast destination address (DA) and the content of Multicast Mask Address High/Low registers (EMACx_MMAHR/EMACx_MMALR).
2. The result is then compared against the content of the Multicast Address High/Low registers (EMACx_MAHR/EMACx_MALR). When there is a match, the multicast destination address is used as an input to the multicast Hash function for further checking. Otherwise, the packet is deemed as a non-IP packet and is accepted.

When EMAC operates in multicast address mode (EMACx_RMR[MAE] = 1), EMAC performs a calculation on the contents of the DA field (logical address filter) as in multiple individual mode, in order to determine whether or not the packet should be accepted.

The logical address filter hardware implements a hash code searching technique commonly used by programmers. The hardware maps the DA of the incoming packet into one of 256 categories corresponding to 256 bits stored in the EMACx_IAHT1–EMACx_IAHT8 or EMACx_GAHT1–EMACx_GAHT8 registers. The hardware accepts or rejects the packet, depending on the state of the corresponding bit in the EMACx_IAHT1–EMACx_IAHT8 or EMACx_GAHT1–EMACx_GAHT8 registers corresponding to the selected category.

Figure 28-15 on page 1019 shows the details of the hardware mapping algorithm. The example depicts multiple individual address mode, but with changes can be used for the multicast address mode.

If the most significant bit (msb) of an incoming address is 0, the address is individual and is passed to the individual address filter. If the msb of an incoming address is 1, the address is multicast and is passed to the multicast address filter. The individual/multicast address filter is a 256-bit mask composed of the EMACx_IAHT1–EMACx_IAHT8 or EMACx_GAHT1–EMACx_GAHT8 registers (each register contains 32 bits of a 256-bit mask). The incoming address is sent through the FCS circuit. After the 48 address bits have gone through the FCS circuit, the high-order eight bits of the resulting FCS (32-bit CRC) are used to select one of the 256 bit positions in the individual/multicast address filter. If the selected filter bit is 1, the address is accepted.

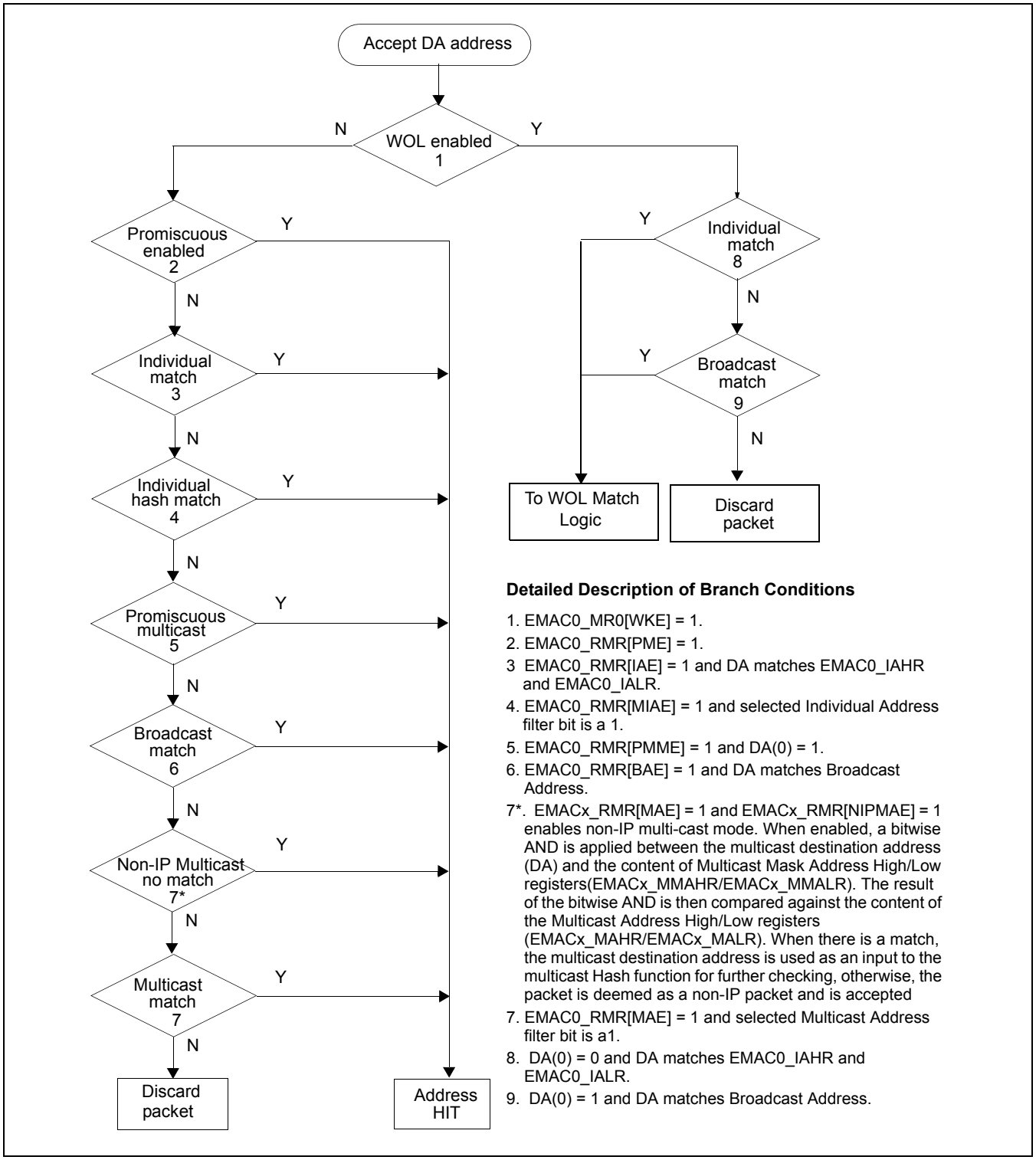
Note: The individual/multicast address filter ensures only that there is a possibility that the incoming packet belongs to this node. To determine if the packet belongs to the node, the incoming individual/multicast address propagated to the main memory is compared by software to the list of logical addresses to be accepted by this node.

For software, the task of mapping an individual/multicast address to one of 256 bit positions requires a program that uses the same CRC algorithm to calculate the hash. See *FCS Calculation* on page 1007 for the 32-bit CRC equation.

28.9.3.2 WOL Mode

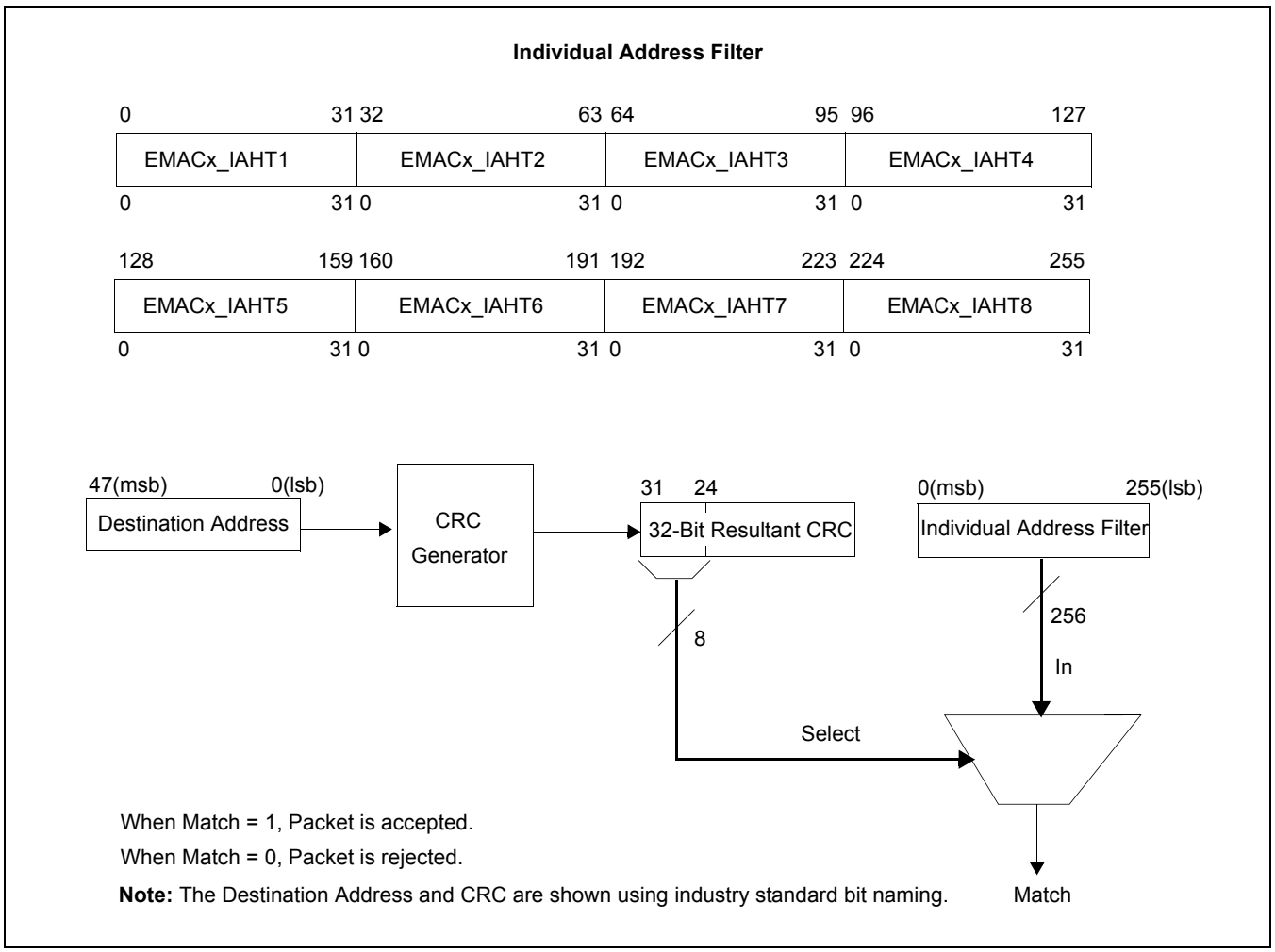
In WOL mode (EMACx_MR0[WKE] = 1), EMAC operates only with the broadcast or individual address in the destination address field.

Figure 28-14. Receive Address Recognition Flowchart



The example in *Figure 28-15* shows that the received individual address maps into category 24, and that bit 8 in EMACx_IAHT2 is set. The match indication is activated and the packet should be accepted.

Figure 28-15. Ethernet Address Filter Operation



28.10 EMAC Registers

In the PPC460EX/EXr/GT there are two EMACs, designated 0 and 1. In the following sections, the registers are specified with a generic name where x represents 0 or 1. EMAC registers are accessed at 0x4 EF60_nnxx where nn = 0E for EMAC0 and nn = 0F for EMAC1.

In the PPC460GT there are four EMACs, designated 0, 1, 2, and 3. In the following sections, the registers are specified with a generic name where x represents 0, 1, 2, or 3. EMAC registers are accessed at 0x4 EF60_nnxx where nn = 0E for EMAC0, nn = 0F for EMAC1, nn = 11 for EMAC2, and nn = 12 for EMAC3.

This section describes the EMAC registers, which are listed in *Table 28-5*. In the register descriptions, the registers are prefixed "EMACx" to denote the identical register implementations in each EMAC. The EMAC registers are accessed using the OPB slave interface. Access to these memory-mapped I/O (MMIO) registers should be word-aligned (4 bytes).

Note: It is necessary to perform a soft reset of the EMAC before initialization by setting EMACx_MR0[SRST] = 1.

Table 28-5. EMACx Register Summary

Mnemonic	Address	Write Access	Power-on Reset Value	Access	Page
EMACx_MR0	0x4 EF60 nn00	See description in *Scenario 1* on page 1054	0xC0000000	R/W	1024
EMACx_MR1	0x4 EF60 nn04	Reset	0x00000000	R/W	1022
EMACx_TMR0	0x4 EF60 nn08	See description on page 1024	0x00000007	R/W	1024
EMACx_TMR1	0x4 EF60 nn0C	See description on page 1024	0x780FC000	R/W	1024
EMACx_RMR	0x4 EF60 nn10	Reset	0x00000007	R/W	1026
EMACx_ISR	0x4 EF60 nn14	Always	0x00000000	R/W	1027
EMACx_ISER	0x4 EF60 nn18	Reset	0x00000000	R/W	1029
EMACx_IAHR	0x4 EF60 nn1C	Reset, R, T	0x00000000	R/W	1031
EMACx_IALR	0x4 EF60 nn20	Reset, R, T	0x00000000	R/W	1032
EMACx_VTPID	0x4 EF60 nn24	Reset, R, T	0x00008808	R/W	1032
EMACx_VTCI	0x4 EF60 nn28	Reset, R, T	0x00000000	R/W	1032
EMACx_PTR	0x4 EF60 nn2C	Reset, T	0x0000FFFF	R/W	1033
EMACx_MAHR	0x4 EF60 nn30	Reset, R, T	0x00000000	R/W	1039
EMACx_MALR	0x4 EF60 nn34	Reset, R, T	0x00000000	R/W	1039
EMACx_MMAHR	0x4 EF60 nn38	Reset, R, T	0x00000000	R/W	1039
EMACx_MMALR	0x4 EF60 nn3C	Reset, R, T	0x00000000	R/W	1039
	0x4 EF60 nn40	Reserved			
	0x4 EF60 nn44	Reserved			
	0x4 EF60 nn48	Reserved			
	0x4 EF60 nn4C	Reserved			
EMACx_LSAH	0x4 EF60 nn50	Not applicable	0x00000000	R	1034
EMACx_LSAL	0x4 EF60 nn54	Not applicable	0x00000000	R	1034
EMACx_IPGVR	0x4 EF60 nn58	Reset, T	0x00000004	R/W	1035
EMACx_STACR	0x4 EF60 nn5C	See description on page 1035	0x00008000	R/W	1035
EMACx_TRTR	0x4 EF60 nn60	See description on page 1036	0x00000000	R/W	1036
EMACx_RWMR	0x4 EF60 nn64	Reset	0x04000800	R/W	1037
EMACx_OCTX	0x4 EF60 nn68	Not applicable	0x00000000	R	1038
EMACx_OCRX	0x4 EF60 nn6C	Not applicable	0x00000000	R	1038
EMACx_IPCR	0x4 EF60 nn70	Not applicable	0x00000000	R/W	1039
EMACx_REVID	0x4 EF60 nn74	Not applicable	0x00000140	R	1038
	0x4 EF60 nn78– 0x4 EF60 nn7C	Reserved			

Table 28-5. EMACx Register Summary (Continued)

Mnemonic	Address	Write Access	Power-on Reset Value	Access	Page
EMACx_IAHT1	0x4 EF60 nn80	Reset, R	0x00000000	R/W	1039
EMACx_IAHT2	0x4 EF60 nn84	Reset, R	0x00000000	R/W	1039
EMACx_IAHT3	0x4 EF60 nn88	Reset, R	0x00000000	R/W	1039
EMACx_IAHT4	0x4 EF60 nn8C	Reset, R	0x00000000	R/W	1039
EMACx_IAHT5	0x4 EF60 nn90	Reset, R	0x00000000	R/W	1039
EMACx_IAHT6	0x4 EF60 nn94	Reset, R	0x00000000	R/W	1039
EMACx_IAHT7	0x4 EF60 nn98	Reset, R	0x00000000	R/W	1039
EMACx_IAHT8	0x4 EF60 nn9C	Reset, R	0x00000000	R/W	1039
EMACx_GAHT1	0x4 EF60 nnA0	Reset, R	0x00000000	R/W	1039
EMACx_GAHT2	0x4 EF60 nnA4	Reset, R	0x00000000	R/W	1039
EMACx_GAHT3	0x4 EF60 nnA8	Reset, R	0x00000000	R/W	1039
EMACx_GAHT4	0x4 EF60 nnAC	Reset, R	0x00000000	R/W	1039
EMACx_GAHT5	0x4 EF60 nnB0	Reset, R	0x00000000	R/W	1039
EMACx_GAHT6	0x4 EF60 nnB4	Reset, R	0x00000000	R/W	1039
EMACx_GAHT7	0x4 EF60 nnB8	Reset, R	0x00000000	R/W	1039
EMACx_GAHT8	0x4 EF60 nnBC	Reset, R	0x00000000	R/W	1039
EMACx_TPC	0x4 EF60 nnC0	Reset, T	0x00000000	R/W	1040

28.10.1 Mode Register 0 (EMACx_MR0)

EMACx_MR0 defines the operating modes of the EMAC, which can be changed at any time during EMAC operation.

Figure 28-16. Mode Register 0 (EMACx_MR0)

0	RXI	Receive MAC Idle 0 RX MAC processing packet 1 RX MAC idle; RX packet processing complete	Read-only
1	TXI	Transmit MAC Idle 0 TX MAC processing packet 1 TX MAC idle; TX packet processing complete	Read-only

2	SRST	EMAC Software Reset 0 EMAC reset is complete 1 Reset the EMAC	Generates a general reset to EMAC through a software command. After setting this bit, EMAC hardware (registers, interface and internal state machines) returns to the power-on reset value. The software writes 1 to this bit in order to drive EMAC to the reset state. The bit is cleared by the hardware when the reset is completed. If EMACx_MR0[SRST] = 1, writing to any EMAC register, and reading any other bit in this register, is not supported. **Note:** The PHY must provide a RX Clk in order to perform a soft reset of the EMAC. If the RX clock is not present, select the internal clock (SDR0_ETH_CFG[EMACx_PHY_CLK_SEL] = 1). After a soft reset, select the external clock.
3	TXE	Transmit MAC Enable 0 TX MAC is disabled 1 TX MAC is enabled	When written to 0, TXMAX is reset.
4	RXE	Receive MAC Enable 0 RX MAC is disabled 1 RX MAC is enabled	When written to 0, RXMAC is reset.
5	WKE	Wake-Up Enable 0 Incoming packets are not examined for wake-up packet 1 Examine incoming packets for wake-up packet	Software can change EMACx_MR0[WKE] only while EMACx_MR0[RXI] = 1 and EMACx_MR0[RXE] = 0.
6:31		Reserved	

28.10.2 Mode Register 1 (EMACx_MR1)

External (to the EMAC) loop-back is achieved by setting EMACx_MR1[FDE] to full-duplex mode and by proper configuration of the attached PHY to activate it's loop-back operational mode.

Internal (to the EMAC) loop-back is achieved by using full-duplex mode and internal loop-back mode simultaneously by setting EMACx_MR1[FDE] = 1 and EMACx_MR1[ILE] = 1. In internal loop-back mode, the EMAC connects the signals output from the TXMAC to the signals that are input to the RXMAC so that transmitted packets are sent to the receive FIFO without activating MII and, therefore, not using a PHY. Full-duplex mode must be used to enable simultaneous transmission and reception without carrier or collision detection.

Figure 28-17. Mode Register 1 (EMACx_MR1)

0	FDE	Full-Duplex Enable 0 Disable simultaneous transmit and receive 1 Enable simultaneous transmit and receive	
1	ILE	Internal Loop-back Enable 0 No wrap back 1 Transmitted packets wrapped back to receive FIFO	Full Duplex must also be set (EMACx_MRI[FDE] = 1).
2	VLE	VLAN Enable 0 Disable processing of VLAN Tags 1 Enable processing of VLAN Tags	

3	EIFC	Enable Integrated Flow Control 0 Disable integrated flow control mechanism 1 Enable integrated flow control mechanism	Refer to *Flow Control* on page 1011 for more details. Set EMACx_MR1[EIFC] = 0 in half-duplex mode.
4	APP	Allow Pause Packet 0 Disables processing of incoming control (pause) packets 1 Enables processing of incoming control (pause) packets	In order for the EMAC to process incoming Pause Packets, the EMACx_RMR [MAE] and EMACx_RMR[BAE] bits must be set to 1.
5:6		Reserved	Always zero
7	IST	Ignore SQE test 0 Wait for end of SQE test period before activation of valid signal 1 Do not wait for end of SQE test period before activation of valid signal	Set EMACx_MR1[IST] = 0 only during half-duplex operation on 10Mbps media.
8:9	MF	Medium Frequency 00 10 Mbps (Ethernet mode) 01 100 Mbps (Fast Ethernet mode) 10 1000 Mbps (Gigabit Ethernet mode) without using the internal GPCS device 11 1000 Mbps (Gigabit Ethernet mode) with the use of internal GPCS device	Defines the operation frequency. MF = 10 is used for GMII and RGMII. MF = 11 is used for SGMII.
10:12	RFS	Receive (RX) FIFO Size 000 512 bytes (32 FIFO entries) 001 1KB (64 FIFO entries) 010 2KB (128 FIFO entries) 011 4KB (256 FIFO entries) 100 8KB (512 FIFO entries) 101 16KB (1024 FIFO entries) 110 Reserved	The highest available settings should always be used. The minimum and maximum values for thresholds discussed in this chapter assume that RFS is set to the maximum available value. Set RFS to 16KB.
13:15	TFS	Transmit (TX) FIFO Size 000 512 bytes (32 FIFO entries) 001 1KB (64 FIFO entries) 010 2KB (128 FIFO entries) - Maximum for EMAC0 and EMAC1 011 4KB (256 FIFO entries) - EMAC2 and EMAC3 (460GT Only) 100 8KB (512 FIFO entries) - EMAC2 and EMAC3 (460GT Only) 101 16KB (1024 FIFO entries) - EMAC2 and EMAC3 (460GT Only) 110 Reserved	The minimum and maximum values for thresholds discussed in this chapter assume that TFS is set to the maximum available values. Set TFS to 2KB in EMAC0 and EMAC1 and to 16KB in EMAC2 and EMAC3 (460GT Only)."
16	TR	Transmit Request 0 Single packet 1 Multiple packets	When TR = 0, Get Next Packet (GNP) is cleared after each packet is sent. GNP must be set for each packet to be transmitted. When TR = 1, GNP is not cleared until all packets currently in the TX FIFO have been transmitted.
17:19	MWSW	Maximum Waiting Status Words 000 A packet is not sent until the status of the previous packet has been received 001 Allows up to one status to be pending when sending the next packet 010 Reserved	Defines the number of status words EMAC can wait for, and still continue transmission. A 0 forces EMAC to wait for every status until requesting a transmission of new packet. For better performance set EMAC0_MR1[MWSW] = 001 to allow pending transfer of one completion status word.
20	JPSM	Jumbo Packet Support Mode 0 Jumbo packet support mode disabled 1 Jumbo packet support mode enabled	Enable only if MF field is set for 1000Mbps. See EMACx_ISR[PTLE] for Jumbo packet definition.
21:25	IPPA	Internal GPCS PHY Address	See GPCS registers in *Gigabit Mode Physical Coding Sublayer (GPCS)* on page 1040.

26:28	OBCI	OPB Bus Clock Indication 000 Divide by 20 (OPBClk = 50 MHz) 001 Divide by 28 (OPBClk = 66.66 MHz) 010 Divide by 34 (OPBClk = 83.33 MHz) 011 Divide by 40 (OPBClk = 100 MHz) 100 Divide by 54 (OPBClk = 100 MHz or greater)	The GMCMDClk is derived from the OPB Clk. This field sets a divisor used to derive the MDIO clock, GMCMDClk, from the OPB clock. GMCMDClk = OPBClk/OBCI divisor GMCMDClk = 50 MHz / 20 = 2.5 MHz GMCMDClk = 66.66 MHz / 28 = 2.38 MHz GMCMDClk = 83.33 MHz / 34 = 2.45 MHz GMCMDClk = 100 MHz / 40 = 2.5 MHz GMCMDClk = 100 MHz / 54 = 1.85 MHz Hold time: • One OPB clock cycle, when OBCI is 0b000, 0b001, 0b010, or 0b011. • Two OPB clock cycles when OBCI is 0b100. Setup time is the MDIO clock cycle minus the hold time. Note: The MDIO clock must be 2.5 MHz or less.
29:31		Reserved	

28.10.3 Transmit Mode Register 0 (EMACx_TMR0)

EMACx_TMR0 defines EMAC operating modes during transmit operations (see *EMAC Transmit Operation* on page 1001).

EMACx_TMR0[GNP] is self-clearing. Writing 0 to these fields has no effect.

Figure 28-18. Transmit Mode Register 0 (EMACx_TMR0)

0	GNP	Get New Packet 0 Writing 0 has no effect. 1 Packet ready for transmission on TX Channel	EMACx_TMR0[GNP] = 0 if EMAC is programmed.
1:28		Reserved	
29:31	TFAE	TX FIFO Almost Empty 000 Reserved 001 Number of used entries ≤ 2 (32 bytes) 010 Number of used entries ≤ 4 (64 bytes) 011 Number of used entries ≤ 8 (128 bytes) 100 Number of used entries ≤ 16 (256 bytes) 101 Number of used entries ≤ 32 (512 bytes) 110 Number of used entries ≤ 64 (1024 bytes) 111 Number of used entries ≤ 128 (2048 bytes)	When the number of occupied entries in the TX FIFO is less than or equal to this limit, and while a packet is transmitted, TX_FIFO_ALMOST_EMPTY interrupt is asserted. **Note:** The value of 111 is applicable only when the Jumbo Packet option is enabled. See *Mode Register 1 (EMACx_MR1)* on page 1022. Otherwise, writing a value of 111 will disable the TX FIFO almost empty limit option, and no interrupt occurs. This is also set as the default option. Each FIFO entry is 16 bytes (the width of a FIFO memory array row).

28.10.4 Transmit Mode Register 1 (EMACx_TMR1)

EMACx_TMR1 shown in *Figure 28-19* defines conditions for activation of MAL service requests during transmit operations (see *EMAC Transmit Operation* on page 1001).

28.10.4.1 Low-Priority Requests

EMAC requests, at low-priority, that MAL begin transferring the next packet's data when the number of vacant TX FIFO entries exceeds the Transmit Low Request (TLR) programmed number of FIFO entries. A FIFO entry is 16 bytes.

MAL service must not be requested until there is room in the TX FIFO for the EOPB burst-length amount of data. Therefore, EMACx_TMR1[TLR] must be programmed to be at least 16 FIFO entries (256 bytes) plus 1 entry when the programmed MAL burst length is the reset initialized value of 64 words (256 bytes) as configured in SDR0_MALxTBL.

Minimum number of TLR entries = (Burst length in bytes/16) + 1

When the MAL burst length = 256, TLRmin $\geq$ 17 entries.

To avoid a transfer deadlock, the sum of EMACx_TMR1[TLR] (specified in FIFO entries) and the TXMAC Transmit Request Threshold (EMACx_TRTR[TRT]) (specified in units of 64 bytes) must be at least four FIFO entries (64 bytes) smaller than the transmit FIFO size, in bytes, specified by EMACx_MR1[TFS].

TLR entries + TRT entries $\leq$ TFS entries $-$ 4
where EMACx_MR1[TFS] is the Transmit FIFO Size in FIFO entries.

28.10.4.2 Urgent-Priority Requests

EMAC requests urgent priority service from MAL if the following conditions occur:

- EMAC begins transmitting the packet to the media before the entire packet is placed in the TX FIFO
- The number of vacant entries available in the TX FIFO exceeds the decimal TUR value

Software must coordinate the value of EMACx_TMR1[TUR] with the value of EMACx_MR1[TFS].The value of EMACx_TMR1[TUR] must be smaller than that of EMACx_MR1[TFS] so that the array address encoded in EMACx_TMR1[TUR] can access the full 66-bit wide array.

The binary value of EMACx_TMR1[TUR] must be greater than that of EMACx_TMR1[TLR].

The EMACx_TMR1 contents can be changed only when EMACx_TMR0[GNP] = 0.

Figure 28-19. Transmit Mode Register 1 (EMACx_TMR1)

0:4	TLR	Transmit Low Request 00000 Reserved 00001 1 FIFO entry (16 bytes) 00010 2 FIFO entries (32 bytes) 00011 3 FIFO entries (48 bytes) . . . 11101 29 FIFO entries (464 bytes) 11110 30 FIFO entries (480 bytes) 11111 30 FIFO entries (480 bytes)	TLR $\geq$ 17 entries
5:7		Reserved	

8:17	TUR	Transmit Urgent Request 0x000 Reserved 0x001 1 FIFO entry (16 bytes) 0x002 2 FIFO entries (32 bytes) 0x003 3 FIFO entries (48 bytes) . . . 0x3FD 1021 FIFO entries (16336 bytes) 0x3FE 1022 FIFO entries (16352 bytes) 0x3FF 1023 FIFO entries (16368 bytes)	TFS > TUR > TLR ≥ 17entries
18:31		Reserved	

28.10.5 Receive Mode Register (EMACx_RMR)

EMACx_RMR defines EMAC operating modes during receive operations.

Figure 28-20. Receive Mode Register (EMACx_RMR)

0	SP	Strip Padding 0 Do not strip pad bytes from the received packet. 1 Strip pad/FCS bytes from the received packet.	
1	SFCS	Strip FCS 0 Do not strip FCS bytes from the received packet. 1 Strip FCS bytes from the received packet.	
2	RRP	Receive Runt Packets 0 Discard packets less than 64 bytes in length. 1 Receive packets less than 64 bytes in length.	
3	RFP	Allow Receive Packets with a FCS Error 0 Discard packets containing a FCS error. 1 Receive packets containing a FCS error.	
4	ROP	Receive Oversize Packet 0 Discard packets that activate Packet Is Too Long error. 1 Receive packets that activate Packet Is Too Long error.	See EMACx_ISR[PTLE] for oversize packet definition.
5	RPIR	Receive Packets with In Range Error 0 Discard packets that activate In Range Error. 1 Receive packets that activate In Range Error.	
6	PPP	Propagate Pause Packet 0 Do not propagate incoming pause packet to MAL; remove packet from FIFO. 1 Propagate incoming pause packet to MAL.	
7	PME	Promiscuous Mode Enable 0 Do not enable promiscuous mode. 1 Accept all packets.	
8	PMME	Promiscuous Multicast Mode Enable 0 Do not accept all multicast packets. 1 Accept all multicast packets.	

9	IAE	Individual Address Enable 0 Do not compare address of received packets with content of individual address register. 1 Compare address of received packets with content of individual address register.	
10	MIAE	Multiple Individual Address Enable 0 Do not compare address of received packets with hash table of individual addresses. 1 Compare address of received packets with hash table of individual addresses.	
11	BAE	Broadcast Address Enable 0 Do not compare address of received packets with broadcast addresses. 1 Compare address of received packets with broadcast addresses.	
12	MAE	Multicast Address Enable 0 Do not compare address of received packets with multicast addresses. 1 Compare address of received packets with multicast addresses.	
13	NIPMAE	Non-IP Multicast Address Enable 0 Do not compare address of received packets with multicast addresses including an additional address match function 1 Compare address of received packets with multicast addresses including an additional address match function.	This bit can only be set to 1only if bit 12 in this register is also set to 1. Otherwise, the EMAC behavior becomes unpredictable.
14:28		Reserved	
29:31	RFAF	RX FIFO Almost Full - IRQ Threshold 000 Reserved 001 Number of used entries ≤ 2 (32 bytes) 010 Number of used entries ≤ 4 (64 bytes) 011 Number of used entries ≤ 8 (128 bytes) 100 Number of used entries ≤ 16 (256 bytes) 101 Number of used entries ≤ 32 (512 bytes) 110 Number of used entries ≤ 64 (1024 bytes) 111 Number of used entries ≤ 128 (2048 bytes)	When the number of occupied entries in the RX FIFO is greater than or equal to this limit, and while a packet is received, RX_FIFO_ALMOST_FULL interrupt is asserted. **Note:** The value of 111 is applicable only when the Jumbo Packet option is enabled. See *Mode Register 1 (EMACx_MR1)* on page 1022. Otherwise, writing a value of 111 will disable the RX FIFO almost full limit option, and no interrupt occurs. This is also set as default option. Each FIFO entry is 16 bytes (the width of a FIFO memory array row).

28.10.6 Interrupt Status Register (EMACx_ISR)

Each EMAC generates a interrupt request (IRQ). The IRQ signal is output from the EMAC into UIC2 interrupt 16 for EMAC0 through UIC2 interrupt 19 for EMAC3. The IRQ is generated based on the content of the Interrupt Status Register EMACx_ISR, ANDed (masked) with the content of the EMACx_ISER. The unmasked Status Register bits are then ORed together to produce one share IRQ signal (per EMAC) that is connected to a UIC input. Therefore, if any of the unmasked conditions occur, an IRQ is generated.

Note: EMAC does not activate its interrupt signal until after the completion-status word for the packet is read by MAL. This occurs when MAL is finished transferring the packet to or from a buffer, with the exception of "MMA Operation Succeed/MMA Operation Failed," which is signaled when the status is known.

A Set status bit is cleared by writing 1 to that bit in the EMACx_ISR. Writing 0 has no effect. The IRQ signal to the UIC is cleared when all Set event indication bits that are in unmasked are cleared.

Figure 28-21. Interrupt Status Register (EMACx_ISR)			
0:1		Reserved	
2	TxPE	Tx Parity Error 0 No Tx parity error 1 Tx parity error occurs when a transmit data bit is damaged at the transmit FIFO	
3	RxPE	Rx Parity Error 0 No Rx parity error 1 Rx parity error occurs when a receive data bit is damaged at the receive FIFO or internal LPRA	
4	TxUE	Tx Underrun Event 0 No underrun event 1 Underrun event while packet is being transferred	Tx FIFO almost empty
5	RxOE	Rx Overrun Event 0 No overrun event 1 Overrun event while packet is being received	Rx FIFO almost full
6	OVR	Overrun 0 No overrun error 1 Overrun error during reception of recent packet	
7	PP	Pause Packet 0 Received packet is not a control pause packet 1 Received packet is a control pause packet	
8	BP	Bad Packet 0 Receive operation OK 1 Early termination was initiated because of a packet error	
9	RP	Runt Packet 0 No Runt packets received 1 Runt packet received	Set when EMACx_RMR[RRP] = 1 and the duration of PHY_RX_DV signal was greater than ShortEventMaxTime constant and less than the collision window.
10	SE	Short Event 0 No short events 1 Duration of PHY_RX_DV signal less than ShortEventMaxTime constant	
11	ALE	Alignment Error 0 No alignment error in received packet 1 Alignment error in received packet	The packet contained an odd number of nibbles (4 bits).
12	BFCS	Bad FCS 0 No FCS error in received packet 1 Packet with an FCS error received	Set if EMACx_RMR[RFP] = 1.
13	PTLE	Packet Too Long Error 0 No oversized packets received 1 Oversized packet received	Set if EMACx_RMR[ROP] = 1 and the received packet length exceeded the maximum allowed value: • 1518 bytes for standard packet (checked only if the length/type field of the transmitted packet contained length value) • 1522 bytes for VLAN tagged packet (checked only if the length/type field of the transmitted packet contained length value and Jumbo support is disabled) • 9018 bytes for Jumbo packet (with no VLAN support) • 9022 bytes for VLAN tagged Jumbo packet

14	ORE	Out Of Range Error 0 Received packet length field value OK 1 Received packet length field value greater than the maximum allowed LLC data size	Indicates that received packet has a length field value greater than the maximum allowed logical link control (LLC) data size (greater than 1500 and less than 1536). The type field contains a value (1501≤ type ≤ 1535) not defined by IEEE or DIX standards.
15	IRE	In Range Error 0 Received packet does not contain an In Range Error 1 Received packet contains an In Range Error	The type field contains a length (type ≤ 1500) that differs from the length of the received frame.
16:22		Reserved	
23	DB	Dead Bit 0 No transmit error or SQE for Tx Channel 1 Transmit error or SQE has occurred for Tx Channel	If EMACx_ISR[DB] = 1, EMAC does not request service for Tx Channel from MAL, even if EMACx_TMR0[GNP] = 1. EMACx_ISR[DB] does not affect EMAC interrupt.
24	SE0	Signal Quality Error 0 No SQEs on Tx Channel 1 SQE test failure during transmission of a packet from Tx Channel	Applicable only in half-duplex mode during 10 Mbps operations; 0 in all other modes.
25	TE0	Transmit Error 0 Tx Channel transmission OK 1 Tx Channel transmission aborted	EMAC aborts the transmitted packet if one of the following events takes place: • Late collision detection • Excessive collision detection • Excessive deferral • TX FIFO underrun • Loss of carrier sense
26:29		Reserved	Always 0
30	MOS	MMA Operation Succeeded 0 MMA_CONTROL addressed on the OPB 1 PHY transfer valid	The device driver should poll assertion of EMACx_ISR[MOS] or EMACx_ISR[MOF] before issuing a new command or before using data read from the PHY.
31	MOF	MMA Operation Failed 0 MMA_CONTROL addressed on the OPB 1 PHY transfer not valid	The device driver should poll assertion of EMACx_ISR[MOF] or EMACx_ISR[MOS] before issuing a new command or before using data read from the PHY.

28.10.7 Interrupt Status Enable Register (EMACx_ISER)

EMACx_ISER indicates which conditions in the EMACx_ISR can generate an interrupt.

Each masking bit in the EMACx_ISER corresponds to a related bit in the EMACx_ISR. If a mask bit is set to 1, the corresponding status bit, when set, causes an interrupt to be generated. Setting a mask bit to 0 suppresses interrupt generation for the associated condition.

Mask bits for reserved bits in the EMACx_ISR are not implemented, have no effect on write, and return 0 on read.

Figure 28-22. Interrupt Status Enable Register (EMACx_ISER)

0:1		Reserved	
2	TxPE	TX Parity Error 0 No Tx parity error 1 Tx parity error occurs when a transmit data bit is damaged at the transmit FIFO	

3	RxPE	RX Parity Error 0 No Rx parity error 1 Rx parity error occurs when a receive data bit is damaged at the receive FIFO or internal LPRA	
4	TxUE	TX Underrun Event 0 No underrun event 1 Underrun event while packet is being transferred	TX FIFO almost empty
5	RxOE	RX Overrun Event 0 No overrun event 1 Overrun event while packet is being received	RX FIFO almost full
6	OVR	Overrun 0 Overrun error will not generate an interrupt. 1 Overrun error will generate an interrupt.	
7	PP	Pause Packet 0 Received control pause packet will not generate an interrupt. 1 Received control pause packet will generate an interrupt.	
8	BP	Bad Packet 0 Early termination on received packet will not generate an interrupt. 1 Early termination on received packet will generate an interrupt.	
9	RP	Runt Packet 0 Received runt packet will not generate an interrupt. 1 Received runt packet will generate an interrupt.	
10	SE	Short Event 0 Short event during receive will not generate an interrupt. 1 Short event during receive will generate an interrupt.	
11	ALE	Alignment Error 0 Alignment error in received packet will not generate an interrupt. 1 Alignment error in received packet will generate an interrupt.	
12	BFCS	Bad FCS 0 FCS error in received packet will not generate an interrupt. 1 FCS error in received packet will generate an interrupt.	
13	PTLE	Packet Too Long Error 0 Oversized packets received will not generate an interrupt. 1 Oversized packet received will generate an interrupt.	
14	ORE	Out Of Range Error 0 Out of range error on received packet will not generate an interrupt. 1 Out of range error on received packet will generate an interrupt.	

15	IRE	In Range Error 0 In range error on received packet will not generate an interrupt. 1 In range error on received packet will generate an interrupt.	
16:23		Reserved	
24	SQE	SQE Error 0 SQE error on Tx Channel will not generate an interrupt. 1 SQE error on Tx Channel will generate an interrupt.	
25	TxE	Transmit Error 0 Tx error on Tx Channel will not generate an interrupt. 1 Tx error on Tx Channel will generate an interrupt.	
26:29		Reserved	
30	MOS	MMA Operation Succeeded 0 Successful MMA Operation with a PHY will not generate an interrupt. 1 Successful MMA Operation with a PHY will generate an interrupt.	
31	MOF	MMA Operation Failed 0 Unsuccessful MMA Operation with a PHY will not generate an interrupt. 1 Unsuccessful MMA Operation with a PHY will generate an interrupt.	

28.10.8 Individual Address High Register (EMACx_IAHR)

EMACx_IAHR contains the high-order halfword of the station unique individual address.

During packet reception, if EMAC is programmed in individual address match mode (EMACx_RMR[IAE] = 1), the contents of EMACx_IAHR are concatenated with the content of EMACx_IALR to form a composite address that is compared with the destination address of the received packet. If addresses match, the packet is transferred to MAL.

During packet transmission, EMACx_IAHR is used in source address inclusion/replacement and as the source address field in the self-assembled control (pause) packet.

Figure 28-23. Individual Address High Register (EMACx_IAHR)

0:15		Reserved	
16:31	HO	Receive and Transmit High Order High-order halfword of the station unique individual address	This field contains bits 0:15 of the destination address (bit 0 is the most significant bit).

28.10.9 Individual Address Low Register (EMACx_IALR)

EMACx_IALR contains the low-order word of the station unique individual address.

During packet reception, EMACx_IALR is compared with the corresponding address bits of the received packet.

During packet transmission, EMACx_IALR is used in source address inclusion/replacement and as the source address field in the self-assembled control (pause) packet.

Figure 28-24. Individual Address Low Register (EMACx_IALR)

0:31	LO	Receive and Transmit Low Order Low-order bits of Receive Individual Address or Transmit Source Address	

28.10.10 VLAN TPID Register (EMACx_VTPID)

EMACx_VTPID contains the value of the VLAN TPID (Tag Protocol Identifier) field.

During packet reception, packet bytes 13 and 14 are compared to the content of this register to check whether the packet is tagged with a VLAN ID.

During packet transmission, EMAC uses EMACx_VTPID when VLAN Tag replacement or VLAN Tag inclusion mode is chosen.

The value of this register must be a Type field (8100).

Figure 28-25. VLAN TPID Register (EMACx_VTPID)

0:15		Reserved	
16:31	VIDT	VLAN ID tag	

28.10.11 VLAN TCI Register (EMACx_VTCI)

EMACx_VTCI contains the value of the VLAN TCI (Tag Control Information) field.

During packet transmission, EMAC uses EMACx_VTCI when VLAN Tag replacement or VLAN Tag inclusion mode is chosen.

Figure 28-26. VLAN TCI Register (EMACx_VTCI)

0:15		Reserved	
16:31	VTCI	VLAN TCI tag	

28.10.12 Pause Timer Register (EMACx_PTR)

EMACx_PTR defines the time period for which the pause function is enabled. EMAC uses EMACx_PTR[TVR] as the timer value field of control (pause) packets (see *Control Packet Transmission* on page 1012). Each bit corresponds to 512 bit times.

Figure 28-27. Pause Timer Register (EMACx_PTR)

0:15		Reserved	
16:31	TVF	Timer Value Field	

28.10.13 Multicast Address High Register (EMACx_MAHR)

The Multicast Address High Register contains the upper 16 bits of the multicast address to be used for address match operation for Non-IP multicast addresses.

Figure 28-28. Multicast Address High Register (EMACx_MAHR)

0:15		Reserved	
16:31	RMA	Receive Multicast Address (high order bits)	

If EMACx_RMR[NIPMAE] is set, the content of this field, concatenated with the content of EMACx_MALR, forms a composite address that is compared with the result of the AND function between the destination address (DA) of the received packet and the content of non-IP Multicast Mask High/Low registers (EMACx_MMAHR and EMACx_MMALR).

If the addresses match, the Multicast Hash function is applied. Otherwise, the packet is assumed to be a non-IP packet and is accepted without further address checks.

Bit 23 set to 1, indicates that the destination address is a multicast (unless all bits in the destination address are ones, which means that the address is broadcast).

28.10.14 Multicast Address Low Register (EMACx_MALR)

The Multicast Address Low Register contains the low 32 bits of the multicast address to be used for address match operation for non-IP multicast addresses.

Figure 28-29. Multicast Address Low Register (EMACx_MALR)

0:31	RMA	Receive Multicast Address (low order bits)	This field contains address bits 16:47 of the destination address (DA). Bit 47 is the lsb.

28.10.15 Multicast Mask Address High Register (EMACx_MMAHR)

The Multicast Mask Address High Register contains the upper 16 bits of the mask to be ANDed with the upper 16 bits of the multicast destination address.

Figure 28-30. Multicast Mask Address High Register (EMACx_MMAHR)			
0:15		Reserved	
16:31	MMA	Multicast Mask Address (high order bits)	

28.10.16 Multicast Mask Address Low Register (EMACx_MMALR)

The Multicast Mask Address Low Register contains the low 32 bits of the mask to be ANDed with the low 32 bits of the multicast destination address.

Figure 28-31. Multicast Mask Address Low Register (EMACx_MMALR)			
0:31	MMA	Multicast Mask Address (low order bits)	

28.10.17 Last Source Address High Register (EMACx_LSAH)

EMACx_LSAH contains the high-order halfword of the source address of the last "good" received packet. The packet is considered to be "good" if EMAC is programmed to provide this packet to MAL.

Figure 28-32. Last Source Address High Register (EMACx_LSAH)			
0:15		Reserved	
16:31	LSAHHW	Last Source Address High-Order Halfword	

28.10.18 Last Source Address Low Register (EMACx_LSAL)

EMACx_LSAL contains the low-order word of the source address of the last "good" received packet. The packet is considered "good" if EMAC is programmed to provide this packet to MAL.

Figure 28-33. Last Source Address Low Register (EMACx_LSAL)			
0:31	LSALW	Last Source Address Low-Order Word	

28.10.19 Inter-Packet Gap Value Register (EMACx_IPGVR)

EMACx_IPGVR[IPG] adjusts the amount of idle time between EMAC's transmission of frames and is designated the Inter-Packet Gap (IPG) or Inter-Frame Gap (IFG). The value in the IPG field is 1/24th of the idle time, in bits, the EMAC has between frames. The unit of measure of the IPG field is 24 bit times. The minimum gap allowed by IEEE 802.3 is 96 bit times. Therefore, the minimum value should be 4, which sets the IPG period to be 96 bit times.

Figure 28-34. inter-Packet Gap Value Register (EMACx_IPGVR)

0:25		Reserved	
26:31	IPG	Inter-Packet Gap	In units of 24 bit times. Minimum value = 4 which sets IPG = 96 bit times.

28.10.20 STA Control Register (EMACx_STACR)

EMACx_STACR controls the GMII Management interface. The software must follow the following steps during access to the EMACx_STACR:

1. EMAC sets EMACx_STACR[OC] = 0 to indicate that the data has been written to the PHY, or the data read from the PHY is valid. The device driver should poll for EMACx_STACR[OC] = 0 before issuing a new command, or before using data read from the PHY.
2. Software writes the EMACx_STACR register fields PHYD, IMS, STAOPC, PADR, and DEVADR as needed for the desired command.
3. Software sets EMACx_STACR[OC] = 1 to start the transaction.
4. Return to step 1.

Figure 28-35. STA Control Register (EMACx_STACR)

0:15	PHYD	PHY data	Data to be sent to the PHY if the command is a write, or data is read from the PHY if the command is a read.
16	OC	MMA Go Bit/Operation Complete 0 Operation completed 1 Transaction started	When set to 1 (by software), the transaction starts. This is a self-clearing bit. EMAC clears this bit when the operation completes (either when the data is written to the PHY or when the data read from the PHY is valid). The device driver should poll this bit for a 0 before issuing a new command or before using data read from the PHY. This bit is set to 0 after a POR, after a soft reset, or when a new transaction is started (software triggers the MMA Go Bit). Writing 0 to this bit has no effect.
17	PHYE	PHY Error 0 Successful read transaction 1 Read transaction was not successful	
18	IMS	Indirect Mode Selection 0 Direct access 1 Indirect access	This bit is set to 1 if the access method is Indirect Mode (according to Clause 45 of IEEE802.3). Setting this bit to 0 enforces the use of the old MIO/MDIO (Clause 22 of IEEE 802.3) access method. This bit affects the use of the STAOPC bit. **Note:** In order to access the internal GPCS, this field must be set to 0.

19:20	STAOPC	STA Opcode Direct access OP codes (IMS = 0): 01 Write 10 Read Indirect access OP codes (IMS = 0): 00 Address 01 Write 10 Read 11 Read Inc.	This field contains the opcode to be used during the MIO/MDIO transaction (see IMS field).
21		Reserved	
22:26	PADR	PHY Command Destination Address	This field contains the address of the PHY intended to receive the command. To access an external PHY, set PADR to the desired address and set EMACx_MR1[IPPA] to an address unused by any external PHY. **Note:** In order to access the internal GPCS, this field must be equal to the programmed internal PCS address (EMACx_MR1[IPPA]).
27:31	DEVADR	PHY Register Address	This field contains the PHY register address.

28.10.21 Transmit Request Threshold Register (EMACx_TRTR)

The Transmit Request-Threshold defined in EMACx_TRTR[TRT] determines the number of bytes that must be placed (by MAL) in the transmit FIFO before a transmission request is made to the Ethernet TXMAC logic. The TXMAC then activates the TX Enable signal to the PHY to indicate the start of frame transmission. Once an entire frame's data is transferred into the transmit FIFO by MAL, TXMAC initiates a transmission even if the transmit request-threshold has not been reached.

EMACx_TRTR can only be written while the TXMAC is idle, which is indicated by EMACx_MR0[TXI] = 1.

Software must ensure that the number of bytes selected for the transmit request-threshold is less the transmit FIFO size, in bytes, specified in the Transmit FIFO size configured by EMACx_MR1[TFS].

To avoid a TX deadlock condition, the sum of the MAL service low-priority request threshold configured in EMACx_TMR1[TLR] (converted into bytes) and the transmit-request threshold configured in EMACx_TRTR[TRT] (in bytes) must be at least four FIFO array entries (64 bytes) smaller than the transmit FIFO size (in bytes) as specified in the Transmit FIFO Size field of EMACx_MR1[TFS].

TLR entries + TRT entries ≤ TFS entries − 4; where EMACx_MR1[TFS] is the Transmit FIFO Size in FIFO entries.

To avoid an TX under-run condition, the value of this threshold must be high enough that sufficient data is the TX FIFO before transmission of a frame is initiated. Typically, setting the threshold so that half of the FIFO is occupied before beginning transmission ensures that underruns do not occur regardless of how busy the MAL and DRAM are. The only trade off of using larger threshold versus smaller ones is that a negligibly small additional amount of latency will be incurred before initiation of transmission of frames which are larger than the threshold— but only when it is the first frame placed into an otherwise empty TX FIFO.

In order to allow retransmission of a frame when a collision occurs, which can only occur, and is only detected in half-duplex mode, EMAC preserves the necessary 64 octets of data in the Transmit FIFO until it gets an indication that the time window for collision detection, which is 64 octets at the media's bit clock rate, has closed.

Figure 28-36. Transmit Request Threshold Register (EMACx_TRTR)			
0:7	TRT	Transmit Request Threshold The following number of bytes must be placed in the Transmit FIFO before initiating a transmit request. 0000_0000 64 bytes (4 FIFO entries) 0000_0001 128 bytes (8 FIFO entries) 0000_0010 192 bytes (12 FIFO entries) 0000_0011 256 bytes (16 FIFO entries) . . . 1111_1100 16192 bytes (1012 FIFO entries) 1111_1101 16256 bytes (1016 FIFO entries) 1111_1110 16320 bytes (1020 FIFO entries) 1111_1111 16384 bytes (1024 FIFO entries)	TRT is in increments of 64 bytes or 4 FIFO entries.
8:31		Reserved	

28.10.22 Receive Low/High Water Mark Register (EMACx_RWMR)

The EMACx_RWMR contains Low and High water marks defining the conditions that cause the EMAC to activate a MAL service request, and determine when a optional Pause Packet is transmitted.

EMAC activates a low priority request for MAL service when the number of occupied entries in the receive FIFO reaches the Receive Low Water Mark, RLWM. Additionally, a request for the optional Pause packet with a pause_value of 0 is issued when the number of empty entries reaches the RLWM.

Software must coordinate the value of EMACx_RWMR[RLWM] with the value of EMACx_MR1[RFS]. EMACx_RWMR[RLWM] should be smaller than EMACx_MR1[RFS] and larger than the MAL burst length.

Notes: A FIFO entry is 16B, the width of a FIFO memory array row which is 128 bits.

In the PPC460EX/EXr/GT, the user can select the MAL burst length for each MAL RX or TX channel by writing to the appropriate fields in the SDR0_MALRBL and SDR0_MALTBL registers.

Once an entire packet is placed in the receive FIFO, EMAC initiates a low priority MAL service request regardless of the settings of the Low or High water mark thresholds.

EMAC activates an urgent priority request if the number of occupied entries in the Receive FIFO is greater than or equal to EMACx_RWMR[RHWM] (the receive high water mark is reached). A request for a pause packet with a programmable pause timer value is also issued when the receive high water mark is reached.

Software must insure the value of EMACx_RWMR[RHWM] is greater than the value of EMACx_RWMR[RLWM] and less than the size, in FIFO entries, of the receive FIFO specified, in bytes, in EMACx_MR1[RFS].

Figure 28-37. Receive Low/High Water Mark Register (EMACx_RWMR)

0:9	RLWM	Receive Low Water Mark 0x000 0 FIFO entries 0x001 1 FIFO entry (16 bytes) 0x002 2 FIFO entry (32 bytes) 0x003 3 FIFO entry (48 bytes) . . . 0x3FD 1021 FIFO entry (16336 bytes) 0x3FE 1022 FIFO entry (16352 bytes) 0x3FF 1023 FIFO entry (16368 bytes)	
10:15		Reserved	
16:25	RHWM	Receive High Water Mark 0x000 0 FIFO entries 0x001 1 FIFO entry (16 bytes) 0x002 2 FIFO entry (32 bytes) 0x003 3 FIFO entry (48 bytes) . . . 0x3FD 1021 FIFO entry (16336 bytes) 0x3FE 1022 FIFO entry (16352 bytes) 0x3FF 1023 FIFO entry (16368 bytes)	
26:31		Reserved	

28.10.23 Transmitted Octets Register (EMACx_OCTX)

The read-only EMACx_OCTX register contains the number of transmitted octets.

Figure 28-38. Transmitted Octets Register (EMACx_OCTX)

0:31	OCTX	Number of octets (bytes) transmitted.	

28.10.24 Received Octets Register (EMACx_OCRX)

The read-only EMACx_OCRX register contains the number of received octets.

Figure 28-39. Received Octets Register (EMACx_OCRX)

0:31	OCRX	Number of octets (bytes) received.	

28.10.25 Internal PCS Configuration Register (EMACx_IPCR)

The read/write EMACx_IPCR register defines various configuration parameters for the internal PCS, Gigabit mode Physical Coding Sublayer (GPCS) unit.

Figure 28-40. Internal PCS Configuration Register (EMACx_IPCR)			
0:21	OUI	OUI value	
22:27	MMN	Manufacturer Model Number	
28:31	REVID	Revision number.	

28.10.26 Revision ID Register (EMACx_REVID)

This register contains the revision number and the branch revision number of the core and is read only.

Figure 28-41. Revision ID Received (EMACx_REVID)			
0:11		Reserved	
12:23		Revision number	
24:31		Branch revision number.	

28.10.27 Individual Address Hash Tables 1–8 (EMACx_IAHT1–EMACx_IAHT8)

These registers are used in the hash table function of the multiple individual addressing mode.

See *Address Match Mechanism* on page 1016 for more information. See *Figure 28-15* on page 1019 for bit mapping information.

Figure 28-42. Individual Address Hash Tables 1–8 (EMACx_IAHT1–EMACx_IAHT8)			
0:31	IAHN	Individual Address Hash Number	

28.10.28 Group Address Hash Tables 1–8 (EMACx_GAHT1–EMACx_GAHT8)

These registers are used in the hash table function of the group addressing mode.

See *Address Match Mechanism* on page 1016 for more information. See *Figure 28-15* on page 1019 for bit mapping information.

Figure 28-43. Group Address Hash Tables 1–8 (EMACx_GAHT1–EMACx_GAHT8)			
0:31	GAHN	Group Address Hash Number	

28.10.29 Transmit Pause Control Register (EMACx_TPC)

A write to this register allows the upper level software to send a pause packet towards the Ethernet media.

Figure 28-44. Transmit Pause Control Register (EMACx_TPC)			
0	IPA	Issue a Pause Packet	High until finished sending a self-generated Pause Packet. Then cleared internally.
1:16	TV	Timer Value	Used for a self-generated pause packet.
17:31		Reserved	

28.10.30 Gigabit Mode Physical Coding Sublayer (GPCS)

The following sections describe the GPCS operation.

Note: Configuration of the GPSC registers is required when using SGMII.

28.10.30.1 GPCS Overview

The Gigabit mode Physical Coding Sublayer (GPCS) unit is an implementation of the Physical Coding Sublayer (PCS) which connects a Media Access Control (MAC) with the Physical Medium Attachment (PMA) sublayer, via the PMA service interface. The PCS, MAC and PMA service interface are defined in the IEEE P802.3z Standard, Clauses 36 and 37.

The GPSC registers are accessed internally through the MII/GMII Management interface by way of the *STA Control Register (EMACx_STACR)* on page 1035. To address the GPCS registers, the EMACx_STACR[PADR] and EMACx_MR1[IPPA] must be set to the same address. The *Internal PCS Configuration Register (EMACx_IPCR)* on page 1039 defines various configuration parameters for GPCS.

Note: All PowerPC registers have bit 0 as the most significant bit while all GPCS 16-bit registers have bit 15 as the most significant bit to be consistent with IEEE specifications.

28.10.30.2 GPSC Registers

The following table provides a summary of the GPSC registers followed by a detailed description of each register.

Table 28-6. GPCS Register Summary

Mnemonic	Register	Address	Access	Page
GPCSx_CR	GPCS Control Register	0x00	R/W	1041
GPCSx_SR	GPCS Status Register	0x01	R	1042
GPCSx_ID0	GPCS ID0 Register	0x02	R	1043
GPCSx_ID1	GPCS ID1 Register	0x03	R	1044
GPCSx_ANAR	GPCS Auto Negotiation Advertisement Register	0x04	R/W	1044
GPCSx_ANLR	GPCS Auto Negotiation Link Partner Base Page Ability Register	0x05	R	1046
GPCSx_ANER	GPCS Auto Negotiation Expansion Register	0x06	R	1047
GPCSx_ANPTR	GPCS Auto Negotiation Next Page Transmit Register	0x07	R/W	1047

Table 28-6. GPCS Register Summary (Continued)

Mnemonic	Register	Address	Access	Page
GPCSx_ANLNPR	GPCS Auto Negotiation Link Partner Ability Next Page Register	0x08	R	1048
GPCSx_ESR	GPCS Extended Status Register	0x0F	R	1048
GPCSx_RAR	GPCS Resolved Ability Register	0x10	R	1049
GPCSx_ISR	GPCS Interrupt Status Register	0x11	R/W	1049
GPCSx_ISER	GPCS Interrupt Status Enable Register	0x12	R	1050
GPCSx_CFG	GPCS Configuration Register	0x13	R/W	1051

28.10.30.3 Recommended GPCS Register Settings for SGMII

After a system or chip reset, perform the following configurations.

1. Enable auto-negotiate or select the port speed.
   - Enable auto-negotiation. Set SDR0_ETH_CFG[SGMII_AUTO]=1, SDR0_ETH_CFG[SGMII0_SPEED_SEL]=0 and SDR0_ETH_CFG[SGMII0_SPEED_SEL]=0 to enable auto-negotiation on SGMII0 and SGMII1 ports.(PPC460EX/EXr/GT rev B only)
   - Disable auto-negotiation and select port speed. Set SDR0_ETH_CFG[SGMII_AUTO]=1 and set SDR0_ETH_CFG[SGMII0_SPEED_SEL] and/or SDR0_ETH_CFG[SGMII0_SPEED_SEL] to the desired speed. (PPC460EX/EXr/GT rev B only)
   - SGMII0 and SGMII1 ports on the PPC460EX/EXr/GT rev A do not support auto-negotiation. See errata.
   - SGMII2 port does not support auto-negotiation. This port only operates at 1Gbps.(PPC460GT only)

   Note: The settings in SDR0_ETH_CFG control auto-negotiation. The settings controlling auto-negotiation in GPCSx_ANER, GPCSx_ANPTR, GPCSx_ANLNPR, GPCSx_ESR and GPCSx_RAR registers are ignored.
2. Enable the SGMII port, SDR0_ETH_CFG[SGMIIx_ENABLE] = 1.
3. Configure the following EMAC registers:
   - Access to the GPCS registers is through the EMACx_STACR register so it is important to set EMACx_STACR[PADR] and EMACx_MR1[IPPA] to the same address.
   - Configuration of EMACx_IPCR is not critical since the SGMII SerDes is internal.
   - Set EMACxMR[MF] = 0b11 to enable the EMAC to communicate with the GPCS.
4. Reset the GPCS by setting GPCSx_CR = 0x8000. (Set GPCSx_CRP[PR] to 1.)
5. Wait until the GPCS exits soft-reset by polling GPCSx_CR[PR]. Once GPCSx_CR[PR] = 0 go to step 5.
6. Set GPSCx_ANAR = 0x8120.
7. Set GPSCx_ANPTR = 0x2801
8. Configure GPSCx_CR.
   - Set GPCSx_CR = 0x1100 to enable auto-negotiate and full duplex.
   - Set GPCSx_CR = 0x0100 to disable auto-negotiate and enable full duplex.

   Note: Only full duplex is supported. The GPCSx_CR[DM] bit field setting is ignored.

28.10.30.4 GPCS Control Register (GPCSx_CR)

The GPCS control register (GPCSx_CR) controls the activation of some of GPCS operation modes including the activation of the renegotiation function by the SMU. *Figure 28-45* describes the GPCSx_CR register bits.

Figure 28-45. GPCS Control Register (GPCSx_CR)			
15	PR	PhyReset	This is a soft reset bit. When this bit is set, the GPCS unit logic is reset, and all GPCS registers go to their default value. While the value of this bit is a logic 1, any attempt to write to the GPCS registers has no effect on the registers. This bit is self-clearing.
14	PL	PhyLoop	When this bit is set, GPCS asserts PHY_LOOP_EN. Typically, PHY_LOOP_EN is connected to the SerDes (such as EWRAP) so that the SerDes can be configured for LoopBack mode. The default value is 0.
13	SS13	SpeedSelection13	Read only. Returns a 0 when read. Indicates GPSC operates at a PHY speed of 1000Mbps.
12	ANE	AutoNegEnable	When this bit is set, Auto-Negotiation is enabled. The default value is 0.
11	PD	PowerDown	When this bit is set, GPCS is placed in Power Saving mode. When the GPCS unit is in Power Saving mode, it responds only to the management transactions. The default value is 0.
10	ISL	Isolate	When this bit is set, GPCS is placed in the Isolation mode. When GPCS is in Isolation mode, it responds only to the management transactions. This bit has the same influence on GPCS power consumption as bit 11, the PowerDown bit in this register. The value is 1 after hard reset and 0 after soft reset.
9	RAN	RestartAutoNegotiation	When this bit is set, Auto-Negotiation is restarted. This bit is self-clearing. If Auto- Negotiation is disabled, then the value of this bit is always a logic and any attempt to write a logic 1 to this bit is ignored. The default value is 0.
8	DM	DuplexMode	If Auto-Negotiation is disabled (AutoNegEnable bit is cleared), this bit is used in order to configure GPCS unit for half- or full-duplex mode. If this bit is set, full-duplex mode is selected. If this bit is cleared, half-duplex mode is enabled. Note that the value of this bit must be set to the same value as EMACx_MR1[FDE]. When Auto-Negotiation is enabled, this bit has no effect. The default value is 0.
7	CT	CollisionTest	When this bit is set, GPCS asserts the COL signal within 512 BT in response to TX_EN assertion, and deasserts the COL within 16 BT in response to TX_EN deassertion. The default value is 0.
6	SS6	SpeedSelection6	Read only. Returns a 1 when read. Indicates GPSC operates at a PHY speed of 1000Mbps.
5:0		Reserved	

28.10.30.5 GPCS Status Register (GPCSx_SR)

The GPCS status register (GPCSx_SR), along with the GPCS extended status register (GPCSx_ESR) describe the GPCS current status. *Figure 28-46* describes the GPCSx_SR register bits.

Figure 28-46. GPCS Status Register (GPCSx_SR)			
15	100T4	100BASE-T4	Return 0 = PHY cannot perform 100BASE-T4.
14	100FD	100BASE-X Full Duplex	Return 0 = PHY cannot perform full duplex 100BASE-X.
13	100HD	100BASE-X Half Duplex	Return 0 = PHY cannot perform half duplex 100BASE-X.
12	10FD	10 Mbps Full Duplex	When this bit is set, Auto-Negotiation is enabled. The default value is 0.
11	10HD	10 Mbps Half Duplex	Return 0 = PHY cannot operate at 10 Mbps in Full Duplex mode.
10	100T2FD	100BASE-T2 Full Duplex	Return 0 = PHY cannot operate at 10 Mbps in Half Duplex mode.
9	100T2HD	100BASE-T2 Half Duplex	Return 0 = PHY cannot perform full duplex 100BASE-T2.
8	ES	Extended Status	Return 1 = Extended status information in Register 15.
7		Reserved	Always 0.
6	MFPS	MF Preamble Suppression	Return 0 = PHY does not use MII management frames.
5	ANC	Auto-Negotiation Complete	When read as 1, indicates that the Auto-Negotiation between the GPCS4 unit and its Link Partner has been completed and the contents of the following registers are valid: • Auto-Negotiation Advertisement Register • Auto-Negotiation Link Partner Base Page Ability Register • Auto-Negotiation Expansion Register GPCS4 unit returns 0 in this bit if bit 12, the AutoNegEnable bit in the GPCS Control Register, is 0. The default value is 0.
4	RF	Remote Fault	When read as 1, indicates that remote fault condition has been detected. This bit will be set by the Auto-Negotiation block function on receipt of a base page with a non-zero Remote Fault field encoding. It will be cleared each time this register is read via the management interface or by PCS reset. In LoopBack mode, this bit is undefined. The default value is 0.
3	ANA	Auto-Negotiation Ability	Return 1 = PHY can perform Auto-Negotiation.
2	LS	Link Status	This bit is set to a logic 1 when the *xmit* flag indicates DATA. When read as 1, indicates that a valid link has been established. The occurrence of a link failure condition clears Link Status bit.
1	JD	Jabber Detect	Always 0
0	EC	Extended Capability	Return 1 = PHY has extended register capabilities.

28.10.30.6 GPCS ID0 Register (GPCSx_ID0)

The GPCS ID0 register (GPCSx_ID0) together with the GPCS ID1 register (GPCSx_ID1) returns the GPCS organizationally unique identifier (OUI), as it was set by the GPCS sideband signals. *Figure 28-47* describes the GPCSx_ID0 register bits.

Figure 28-47. GPCS ID0 Register (GPCSx_ID0)			
15:0	OUI	OUI(3:18)	The value of these bits adhere to the PHY_OUI(3:18) sideband signals. Bit 15, the OUI(3) bit of this register, corresponds to the PHY_OUI(3) sideband signal, and bit 0, the OUI(18) bit of this register, to the PHY_OUI(18) sideband signal.

28.10.30.7 GPCS ID1 Register (GPCSx_ID1)

The GPCS ID1 register (GPCSx_ID1), along with GPCS_ID0 register (GPCSx_ID0), returns the GPCS OUI, and the GPCS manufacturer model number and revision number, as they were set by the GPCS unit sideband signals. *Figure 28-48* describes the GPCSx_ID1 register bits.

Figure 28-48. GPCS ID1 Register (GPCSx_ID1)			
15:10	OUI	OUI19:24	The value of these bits adhere to the PHY_OUI19:24 sideband signals. Bit 15, the OUI19 bit of this register, corresponds to the PHY_OUI19 sideband signal, and bit 10, the OUI24 bit of this register, to the PHY_OUI24 sideband signal.
9:4	MN	ModeNumber5:0	Manufacturer model number, bits 5:0. Bit 9 of this register corresponds to the ModelNumber 5 bit and bit 0 to the ModelNumber 0 bit. The value of these bits will adhere to the PHY_MODEL_NUMB5:0 sideband signals.
3:0	RN	RevNumber3:0	Revision number, bits 3:0. Bit 3 of this register corresponds to the RevNumber3 bit and bit 0 to the RevNumber0 bit. The value of these bits will adhere to the sideband PHY_REV_NUMB3:0 signals.

28.10.30.8 GPCS Auto Negotiation Advertisement Register (GPCSx_ANAR)

The GPCS auto negotiation advertisement register (GPCSx_ANAR) provides the auto-negotiation block function to advertise the EMAC and GPCS unit capabilities to the link partner. *Figure 28-49* describes the GPCSx_ANAR register bits.

Figure 28-49. GPCS Auto Negotiation Advertisement Register (GPCSx_ANAR)			
15	NP	NextPage	This bit shall be set to logic 1 in order to request next page transmission. Subsequent next pages may set the NextPage bit to logic 0 in order to communicate that there is no more next page information to be sent. The default value is 0.
14		Reserved	Write as 0, ignore on read.

13	RF2	Remote Fault 2	This bit, in combination with bit 12, RF1, denotes the remote fault bits. The encoding is as follows: RF1 RF2 0 0 No error, link OK 0 1 Off-line 1 0 Link _Failure 1 1 Auto-Negotiation Error The default value is 00.
12	RF1	Remote Fault 1	This bit, in combination with bit 13, RF2, denotes the remote fault bits. The encoding is as follows: RF1 RF2 0 0 No error, link OK 0 1 Off-line 1 0 Link _Failure 1 1 Auto-Negotiation Error The default value is 00.
11:9		Reserved	Always 0
8	PS2	Pause 2	This bit, in combination with bit PS1, indicates the pause capability. The encoding is as follows: PS1 PS2 0 0 No Pause 0 1 Asymmetric Pause towards Link Partner 1 0 Symmetric Pause 1 1 Both Symmetric Pause and Asymmetric Pause towards Link Partner The default value is '00
7	PS1	Pause 1	This bit, in combination with bit PS2, indicates the pause capability. The encoding is as follows: PS1 PS2 0 0 No Pause 0 1 Asymmetric Pause towards Link Partner 1 0 Symmetric Pause 1 1 Both Symmetric Pause and Asymmetric Pause towards Link Partner The default value is '00
6	HD	Half Duplex	When this bit is set, the device is capable of operating in Half Duplex mode. The default value is 0. Please note that in case Auto-Negotiation is enabled, this bit must match the EMACx_MR1[FDE] setting.
5	FD	Full Duplex	When this bit is set, the device is capable of operating in Full Duplex mode. The default value is 0. Please note that in case Auto-Negotiation is enabled, this bit must match the EMACx_MR1[FDE] setting.
4:0			Always 0

28.10.30.9 GPCS Auto Negotiation Link Partner Base Page Ability Register (GPCSx_ANLR)

The GPCS auto negotiation link partner base page ability register (GPCSx_ANLR) provides auto negotiation block function to store the link partner abilities. GPCSx_ANLR is read-only register, and the GPCS uses it to combine with the link partner abilities. *Figure 28-50* describes the GPCSx_ANLR register bits.

Figure 28-50. GPCS Auto Negotiation Link Partner Base Page Ability Register (GPCSx_ANLR)

15	NP	NextPage	When this bit is set to 1, the next page transmission is requested. The default value is 0.
14	AKN	Aknowledge	When this bit is set, the Link Partner has successfully received at least three consecutive and matching rx_Config_Reg<D15:D0> values (ignoring the Acknowledge bit value). The default value is 0.
13	RF2	Remote Fault 2	This bit, in combination with bit 12, RF1, denotes the remote fault bits. The encoding is as follows: RF1 RF2 0 0 No error, link OK 0 1 Off-line 1 0 Link _Failure 1 1 Auto-Negotiation Error The default value is 00.
12	RF1	Remote Fault 1	This bit, in combination with bit 13, RF2, denotes the remote fault bits. The encoding is as follows: RF1 RF2 0 0 No error, link OK 0 1 Off-line 1 0 Link _Failure 1 1 Auto-Negotiation Error The default value is 00.
11:9		Reserved	Always 0
8	PS2	Pause 2	This bit, in combination with bit PS1, indicates the pause capability. The encoding is as follows: PS1 PS2 0 0 No Pause 0 1 Asymmetric Pause towards Link Partner 1 0 Symmetric Pause 1 1 Both Symmetric Pause and Asymmetric Pause towards Link Partner The default value is 00
7	PS1	Pause 1	This bit, in combination with bit PS2, indicates the pause capability. The encoding is as follows: PS1 PS2 0 0 No Pause 0 1 Asymmetric Pause towards Link Partner 1 0 Symmetric Pause 1 1 Both Symmetric Pause and Asymmetric Pause towards Link Partner The default value is 00
6	HD	Half Duplex	When this bit is set, the device is capable of operating in Half Duplex mode. The default value is 0.

5	FD	Full Duplex	When this bit is set, the device is capable of operating in Full Duplex mode. The default value is 0.
4:0			Always 0

28.10.30.10 GPCS Auto Negotiation Expansion Register (GPCSx_ANER)

GPCS auto negotiation expansion register (GPCSx_ANER) is used by the auto-negotiation block function to indicate that a new base page has been written to the auto-negotiation link partner base page ability register. *Figure 28-51* describes the GPCSx_ANER register bits.

Figure 28-51. GPCS Auto Negotiation Expansion Register (GPCSx_ANER)

15:3		Reserved	Always 0
2	NP	Next Page Able	Indicates that the GPCS unit has the ability to engage in Next Page transactions. Always set to 1.
1	AKN	Page Received	When this bit is set, a new page has been received and stored in the applicable Auto-Negotiation Link Partner Ability register (Base Page or Next Page). It is reset each time this register is read via the Management interface or by PCS reset. The default value is 0.
0		Reserved	Always 0

28.10.30.11 GPCS Auto Negotiation Next Page Transmit Register (GPCSx_ANPTR)

GPCS auto negotiation next page transmit register (GPCSx_ANPTR) contains the next page link code word to be transmitted when the next page ability is enabled. *Figure 28-52* describes the GPCSx_ANPTR register bits.

Figure 28-52. GPCS Auto Negotiation Next Page Transmit Register (GPCSx_ANPTR)

15	NP	Next Page Enable 0 Last page 1 Additional Next Page(s) will follow	Used by the Next Page function to indicate whether or not this is the last Next Page to be transmitted.
14		Reserved	Write as 0. Ignore on read
13	MP	Message Page 0 Unformatted Page 1 Message Page (default value)	Used by the Next Page function to differentiate a Message Page from an Unformatted Page.
12	AKN2	Acknowledge 2 0 Cannot comply with message (default value) 1 Will comply with message	Used by the Next Page function to indicate that a device has the ability to comply with the message.
11	TOG	Toggle	Used to ensure synchronization with the Link Partner during Next Page exchange. This bit shall always take the opposite value of the Toggle bit in the previously exchanged Link Code Word. The initial value of the Toggle bit in the first Next Page transmitted is the inverse of bit 11 in the base Link Code Word.
10:0	MUC	Message Unformatted Code	Used by the Next Page function to carry a single predefined Message Code. The default value is 00000000001.

28.10.30.12 GPCS Auto Negotiation Next Page Transmit Register (GPCSx_ANLNPR)

GPCS auto negotiation next page transmit register (GPCSx_ANLNPR) is used to store link partner next pages. *Figure 28-53* describes the GPCSx_ANLNPR register bits.

Figure 28-53. GPCS Auto Negotiation Next Page Transmit Register (GPCSx_ANLNPR)

15	NP	Next Page Enable 0 Last page 1 Additional Next Page(s) will follow	Used by the Next Page function to indicate whether or not this is the last Next Page to be transmitted.
14		Reserved	Write as 0. Ignore on read
13	MP	Message Page 0 Unformatted Page 1 Message Page (default value)	Used by the Next Page function to differentiate a Message Page from an Unformatted Page.
12	AKN2	Acknowledge 2 0 Cannot comply with message (default value) 1 Will comply with message	Used by the Next Page function to indicate that a device has the ability to comply with the message.
11	TOG	Toggle	Used to ensure synchronization with the Link Partner during Next Page exchange. This bit shall always take the opposite value of the Toggle bit in the previously exchanged Link Code Word. The initial value of the Toggle bit in the first Next Page transmitted is the inverse of bit 11 in the base Link Code Word.
10:0	MUC	Message Unformatted Code	Used by the Next Page function to carry a single predefined Message Code. The default value is 00000000001.

28.10.30.13 GPCS Extended Status Register (GPCSx_ESR)

The GPCS extended status register (GPCSx_ESR), along with the GPCS status register (GPCSx_ESR), describes the GPCS current status. *Figure 28-54* describes the GPCSx_ESR register bits.

Figure 28-54. GPCS Extended Status Register (GPCSx_ESR)

15	XFD	1000BASE-X Full Duplex	When read as 1, indicates that the PHY has the ability to perform full duplex link transmission and reception. The value of this bit will be adhered to by the PHY_FD sideband signal.
14	XHD	1000BASE-X Half Duplex	When read as 1, indicates that the PHY has the ability to perform half duplex link transmission and reception. The value of this bit will be adhered to by the PHY_HD sideband signal.
13	TFD	1000BASE-T Full Duplex	Always 0
12	THD	1000BASE-T Half Duplex	Always 0
11:0		Reserved	Always 0

28.10.30.14 GPCS Resolved Ability Register (GPCSx_RAR)

The GPCS resolved ability register (GPCSx_RAR) contains the selected operation modes after the auto negotiation block function has resolved the operation modes using the priority resolution function. *Figure 28-55* describes the GPCSx_RAR register bits.

Figure 28-55. GPCS Resolved Ability Register (GPCSx_RAR)

15:9		Reserved	Always 0
8	PS2	Pause 2	This bit reflects the corresponding bit in the Link Partner ability base page register. The default value is 0.
7	PS1	Pause 1	Pause, bit PS1. This bit reflects the corresponding bit in the Auto-Negotiation Link Partner Base Page Ability register. The default value is 0.
6	HD	Half Duplex	If this bit is set, the local device will operate in Half Duplex mode.
5	FD	Full Duplex	If this bit is set, the local device will operate in Full Duplex mode.
4:0		Reserved	Always 0

28.10.30.15 GPCS Interrupt Status Register (GPCSx_ISR)

The GPCS interrupt status register (GPCSx_ISR) is a Read/Write_One_for_Reset register (R/WOR). Writing to the interrupt status register can be done only to reset an interrupt cause, and this is done by writing 1 to the interrupt cause. Writing 0 to a bit in the Interrupt Status register will have no effect on the bit. Writing 1 to a bit in the Interrupt Status register which contains a value of 0, will have no effect on the bit.

Note: It is important to disable LP_NO_Response, LP_Zeroes_Response, and AN_Exceed_LTP via the Interrupt Status Enable register, so that these bits will not cause an interrupt. This is due to unique hardware functionality.

Figure 28-56. GPCS Interrupt Status Register (GPCSx_ISR)

15	SLSTP	Synchronization Loss for Short Time Period. 0 Synchronization loss has not occurred 1 Synchronization loss has occurred	Reset value 0.
14	SLELT	Synchronization Loss Exceeds Link Timer Period 0 Synchronization loss does not exceed link timer period 1 Synchronization loss exceeds link timer period	Reset value 0.
13	ICG	Invalid code-group received 0 GPCS unit has not received an invalid code-group 1 GPCS unit has received an invalid code-group	When Auto-Negotiation is enabled, receiving an invalid code-group causes the GPCS unit to renegotiate. Reset value 0.
12:11		Reserved	Always 0
10	LPNR	Link Partner No Response 0 Link partner responds to GPCS negotiation attempts 1 Link partner does not respond to GPCS negotiation attempts	Reset value 0.

9	LPZR	Link Partner Zeroes Response 0 Link partner zeroes response duration is shorter than the GPCS unit link timer period. 1 Link partner zeroes response duration is greater than the GPCS unit link timer period.	Reset value 0.
8	ANLT	Auto Negotiation Exceeds Link Timer Period 0 Auto negotiation does not exceed link timer period 1 Auto negotiation exceeds link timer period	When GPCS0_ISR[ANLT] = 1 auto-negotiation state machine is non-responsive for a time duration greater than the link timer period. Reset value 0.
7	ANPR	Auto Negotiation Resolve Priority Error 0 Auto negotiation resolve priority error not detected 1 Auto negotiation resolve priority error detected	Auto-Negotiation Resolve Priority Error recognized by the GPCS unit. GPCS unit cannot resolve abilities with its Link Partner. Reset value 0.
6:3		Reserved	Always 0
2	LPOL	Link Partner is off-line. 0 Link partner is not off-line 1 Link partner is off-line	Reset value 0.
1	LPLF	Link Partner Link Failure 0 Link partner link successful 1 Link partner detected link failure	Reset value 0.
0	LPER	Link Partner Auto Negotiation Error 0 Link partner auto negotiation error has not occured 1 Link partner auto negotiation error has occurred	When GPCS0_ISR[LPER] = 1 Link Partner advertises that it cannot resolve abilities with the GPCS unit. Reset value 0.

28.10.30.16 GPCS Interrupt Status Enable Register (GPCSx_ISER)

Note: It is important to disable LP_NO_Response, LP_Zeroes_Response, and AN_Exceed_LTP via the Interrupt Status Enable register, so that these bits will not cause an interrupt. This is due to unique hardware functionality.

Figure 28-57. GPCS Interrupt Status Enable Register (GPCSx_ISER)

15:0	ISER	ISER	Each bit in the Interrupt Status Enable register corresponds to an associated bit in the Interrupt Status register. Status Enable bits in the Interrupt Status Enable register for reserved Status bits in the Interrupt Status Register are not implemented. These bits have no effect on write and return a logic 0 on read. Reset value 0.

28.10.30.17 GPCS Configuration Register (GPCSx_CFG)

GPCS configuration register (GPCSx_CFG) is used by the SMU to configure the GPCS unit to desirable operation modes as follows:

- Activate a renegotiation function prior to changing the operation mode.
- Operate in different Link Timer periods.

GPCSx_CFG also controls the link timer period to one of four given periods.

Note: The contents of the GPCSx_CFG can be changed only while the GPCS unit is in sleep mode. An attempt to do so at any other time can result in unexpected GPCS unit behavior.

Figure 28-58 describes the GPCSx_CFG register bits.

Figure 28-58. GPCS Configuration Register (GPCSx_CFG)

15:12		Reserved	
11	PDN	Power Down Negotiation 0 GPCS does not renegotiate before powering-down or isolating from GMII 1 GPCS renegotiates to announce to the Link Partner that it is going off-line before powering-down or isolating from GMII.	Reset value is 0.
10	LMN	Loop Mode Negotiation 0 GPCS does not renegotiate before changing to the LoopBack mode. 1 GPCS renegotiates to announce to the Link Partner that it is going off-line before changing to the LoopBack mode	Reset value is 0.
9:7		Reserved	Read as 0
6:5	LTP	Link Timer Period	The value of these bits determines the period of the GPCS Link Timer. bit 6 bit 5 Link Timer Period 0 0 10 ms (+ 10 ms, -0 ms) 0 1 5 ms (+ 5 ms, -0 ms) 1 0 1 ms (+ 1 ms, -0 ms) 1 1 1 us (+ 1 us, -0 us) The reset value is 00
4:0		Reserved	Read as 0

28.11 MII/GMII Interface

EMAC implements all MII functionality in accordance with Clause 22 in the IEEE Std. 802.3u (when operating in 10/100 Mbps media) and all GMII interface functionality in accordance with Clause 35 in the IEEE Std. 802.3z (when operating in 1000 Mbps media).

The MII/GMII interface is a reconciliation sublayer interface which allows a variety of PHYs to be attached to the EMAC Ethernet MAC without future upgrade problems.

28.11.1 MII Station Management Interface

The EMAC MII station management unit (STA) implements a specific protocol and a special packet format to exchange management packets with the registers of the attached PHY. EMAC automatically generates MII management packets, which conform to Clause 22 in IEEE Std. 802.3u. The EMAC uses the EMACx_STACR for generation of the management packet.

28.12 Programming Notes

When writing software to implement initialization and device drivers, ensure that the following guidelines are used. A FIFO Entry for EMAC is 128 bits or 16 bytes. A FIFO entry for EMAC is 128-bits wide, the width of a row of the memory used for the FIFO memory arrays.

28.12.1 Transmit Threshold Considerations

The TXMAC Transmit Low-priority Request threshold for MAL service is controlled by the EMAC Transmit Mode Register 1 in EMACx_TMR1[TLR]. TLR must be set to a value large enough to ensure a MAL service request is not made to place more data into the TX FIFO until the TX FIFO has sufficient empty space to hold the number of bytes transferred by the MAL in a MAL data transfer-burst transaction. The number of bytes in a MAL data transfer burst is configured for Transmit channels in SDR0_MALTBL. The Reset initialized MAL burst length is 256 bytes.

The TXMAC Transmit Urgent-priority Request threshold value configured in EMACx_TMR1[TUR] must be set to a value or equal to or normally, greater than, the value placed in EMACx_TMR1[TLR]. It must also be a value which specifies a threshold which is less than the Transmit FIFO size as specified in EMACx_MR1[TFS].

To avoid deadlock, the sum of low-priority request set in EMACx_TMR1[TLR] (converted to bytes) and the transmit request threshold set in EMACx_TRTR[TRT] (converted to bytes), must be at least four FIFO entries (64 bytes) less than the Transmit FIFO size specified (in bytes) in EMACx_MR1[TFS].

Transmit Threshold Considerations:

- TLR $\geq$ 17 entries
- TLR + TRT $<$ TFS $-$ 4 entries

Example threshold settings assuming maximum TRT:

- TLR + TRT $<$ TFS $-$ 4
- 17 + TRT $<$ TFS $-$ 4
- TRT $<$ TFS $-$ 4 $-$ 17
- TRT $<$ TFS $-$ 21 FIFO entries
- TRTmax $<$ TFS $-$ 21 $-$ 1
- TRTmax $<$ TFS $-$ 22 FIFO entries
- Transmit FIFO size: TFS = 16KB or 1024 FIFO entries.
- TRTmax $<$ TFS $-$ 22 FIFO entries
- TRTmax $<$ 1024 entries $-$ 22 FIFO entries = 1002 FIFO entries

28.12.2 Receive Watermark Considerations

EMACx_RWMR[RLWM] must be greater than the MAL burst length when specified in FIFO entries. The default burst length size is 256 bytes (16 FIFO entries), SDR0_MALRBL[RBLx] = 0b101.

EMACx_RWMR[RHWM] must be greater than EMACx_RWMR[RLWM].

EMACx_RWMR[RHWM] must be less than the Receive FIFO size in entities (EMACx_MR1[RFS]).

Receive Watermark Constraints:

- 16 FIFO entries ≤ RLWM < RHWM < RFS
- 16 FIFO entries ≤ RLWM < RHWM < 1024 FIFO entries; when RFS of 16KB FIFO corresponds to 1024 FIFO entries.

The following values are a good starting point. Some tuning of these values might be needed depending on the application:

- Set RLWM = 16 entries
- Set RHWM = 768 entries

28.12.3 Other Configuration Considerations

In half-duplex mode selected by setting EMACx_MR1[FDE] = 0; when using Gigabit Ethernet mode, that is, EMACx_MR1[MF] = 10, the transmit FIFO size defined in EMACx_MR1[TFS] must be greater than 512 bytes to avoid transmit underrun errors.

When internal loopback is enabled by setting EMACx_MR1[ILE] = 1, EMAC must be configured in full-duplex mode by setting EMACx_MR1[FDE] = 1.

If EMACx_MR1[MF] = 11 (the internal GPCS is used), the internal GPCS must be reset by setting GPCSx_CR = 0x8000.

28.12.4 Reset and Initialization

The EMAC must be initialized after a reset, or before performing configuration changes. The following types of reset operations can be applied to EMAC.

- Hard Reset. When RESET input is asserted, EMAC aborts all on-going activities unconditionally, initializes all internal state machines, counters, registers, and flushes transmit and receive FIFOs. To be recognized, the reset signal must be asserted for at least two cycles of the slowest clock domain inside EMAC (indicating that the hard reset must be at least 800 ns).
- Soft Reset. Software first should reset the appropriate MAL channels and then begin a soft reset by setting EMACx_MR0[SRST] = 1. In response to the soft reset, EMAC aborts all on-going activities unconditionally, initializes all internal state machines, counters, registers, and flushes transmit and receive FIFOs. After EMAC finishes all activities related to the soft reset processing, EMACx_MR0[SRST] = 0.
- Smart Reset. The software initializes smart reset mode by writing 0 to EMACx_MR0[TXE] or EMACx_MR0[RXE], or to both. In this case, the Ethernet MAC sub-block completes on-going activity (receive, transmit, or both) and then goes to the related Idle state (indicated by setting either EMACx_MR0[TXI] = 1 or EMACx_MR0[RXI] = 1, or both). In this case, the control logic sub-block of EMAC is still accessible for OPB and MAL transactions.

Before performing the necessary configuration changes in EMAC, the software must follow one of the following scenarios. Then the EMAC can be properly configured.

28.12.4.1 Scenario 1

- Hard/soft reset was activated.
- During hard/soft reset, EMACx_MR0[TXE] and EMACx_MR0[RXE] are reset.
- Software detects that the EMACx_MR0[SRST] is reset (after soft reset only).
- Software keeps EMACx_TMR0[GNP] = 0.
- The software can change one or more fields in registers marked with a Reset write access mode in *Table 28-5 EMACx Register Summary* on page 1020 (actually, all EMAC registers are accessible in this scenario).
- The software initializes EMACx_TMR0[GNP] as appropriate.
- The software configures EMACx_MR0[TXE, RXE].

28.12.4.2 Scenario 2

- Software sets EMACx_MR1[EIFC] = 0.
- Software sets EMACx_MR0[TXE] = 0.
- The TXMAC component of the Ethernet MAC sub-block completes on-going activity and then sets EMACx_MR0[TXI] = 1 to enter the related Idle state.
- Software detects EMACx_MR0[TXI] = 1.
- Software performs the necessary EMAC configuration, keeping EMACx_MR0[TXE] = 0. The software can access only part of the EMAC registers marked with write access mode T in *Table 28-5 EMACx Register Summary* on page 1020.
- After all configuration is done, software can set EMACx_MR0[TXE] = 1.

Note: When Scenario 2 occurs, EMAC can still receive packets if EMACx_MR0[RXE] = 1. Scenarios 2 and 3 can occur simultaneously.

28.12.4.3 Scenario 3

- Software sets EMACx_MR0[RXE] = 0.
- The RXMAC component of the Ethernet MAC sub-block completes on-going activity and then sets EMACx_MR0[RXI] = 1 to enter the related Idle state.
- Software detects EMACx_MR0[RXI] = 1.
- Software performs the necessary EMAC configuration, keeping EMACx_MR0[RXE] = 0. The software can access only part of EMAC registers marked with write access mode R in *Table 28-5 EMACx Register Summary* on page 1020.
- After all configuration is done, software can set EMACx_MR0[RXE] = 1.

Note: When Scenario 3 occurs, EMAC can still transmit packets if EMACx_MR0[TXE] = 1. Scenarios 2 and 3 can occur simultaneously.

29. EMAC to PHY Interface Bridges

The PPC460EX/EXr/GT provides three bridges called the ZMII, RGMII, and MDIO bridges that connect the Ethernet media access controllers (EMACs) to standard physical devices (PHYs). This chapter lists the various Ethernet combinations and provides detailed information on ZMII, RGMII, and MDIO bridge configurations.

The Ethernet controllers have interfaces available to communicate with a PHY. For both EMACs, which support 10/100/1000 Mbps mode, there is one MII (media independent interface). For gigabit (1000 Mbps) speeds one controller can use the full GMII (gigabit MII), or each controller can use RGMII (reduced GMII).

Figure 29-1. ZMII and RGMII Bridges and SGMII SerDes Configurations

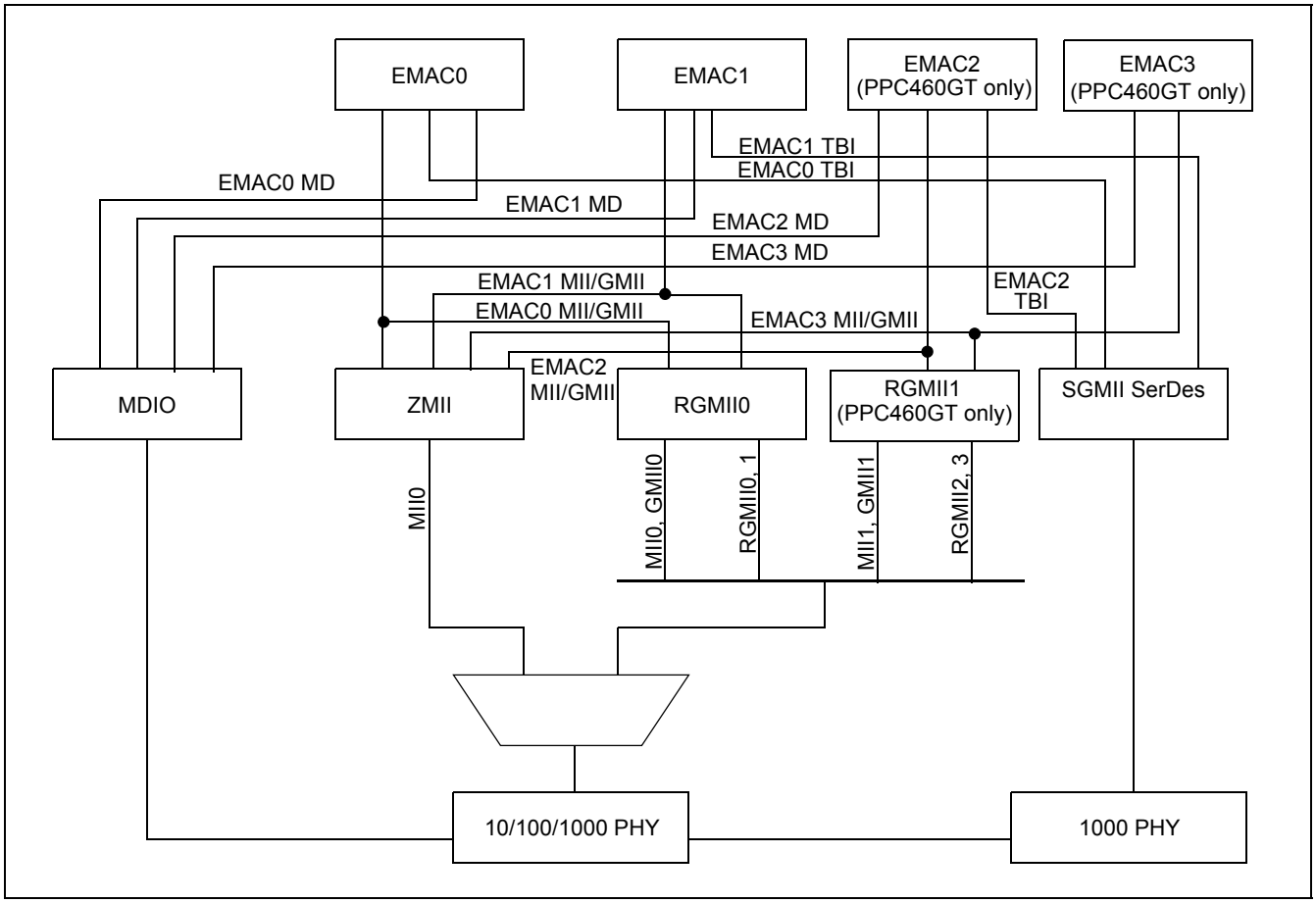


Table 29-1 lists the Ethernet interface combinations supported by the PPC460EX.

Table 29-1. PPC460EX Ethernet Interfaces

Configuration	EMAC0*	EMAC1*
1	RGMII0	RGMII1
2	RGMII0	SGMII1
3	GMII0 or MII0	SGMII1
4	SGMII0	SGMII1
*EMAC0 and EMAC1 have QoS and TAH support.		

Table 29-2 lists the Ethernet interface combinations supported by the PPC460EXr.

Table 29-2. PPC460EXr Ethernet Interfaces

Configuration	EMAC0*	EMAC1*
1	RGMII0	RGMII1
2	GMII0 or MII0	SGMII1
*EMAC0 and EMAC1 have QoS and TAH support.		

Table 29-3 lists the Ethernet interface combinations supported by the PPC460GT.

Table 29-3. PPC460GT Ethernet Interfaces

Configurations	EMAC0*	EMAC1*	EMAC2	EMAC3
1	GMII0 or MII0	-	GMII1 or MII1	-
2	RGMII0	RGMII1	RGMII2	RGMII3
3	SGMII0	SGMII1	SGMII2	-
4	GMII0 or MII0	SGMII1	-	-
5	RGMII0	SGMII1	-	-
6	GMII0 or MII0	SGMII1	RGMII2	-
7	RGMII0	SGMII1	GMII1 or MII1	-
8	RGMII0	RGMII1	SGMII2	-
*EMAC0 and EMAC1 have QoS and TAH support.				

Note: TAH logic is selected by:

SDR0_ETH_CFG[TAHOE0_BYPASS]
SDR0_ETH_CFG[TAHOE1_BYPASS]

Table 29-4. GMII/MII Configuration

GMII/MII Interface	Configuration
MII0	MII through the ZMII bridge: EMAC0<—>ZMII Bridge 0 <—>MII0 When using the ZMII bridge, GMCRefClk is not required. SDR0_ETH_CFG[SGMII0_LPBK] = 0 SDR0_ETH_CFG[SGMII0_ENABLE] = 0 SDR0_ETH_CFG[ZMIM] = 0b00 SDR0_ETH_CFG[GMC0BS] = 1 ZMII0_FER[MII0] = 1 EMAC0_MR1[MF] = 0b00 for 10Mbps EMAC0_MR1[MF] = 0b01 for 100Mbps
GMII0/MII0	GMII/MII through the RGMII bridge: EMAC0<—>RGMII Bridge 0<—>GMII0/MII0 GMCRefClk is required. SDR0_ETH_CFG[SGMII0_LPBK] = 0 SDR0_ETH_CFG[SGMII0_ENABLE] = 0 SDR0_ETH_CFG[GMC0BS] = 0 RGMII0_FER[CH0CFG] = 0b111 RGMII0_FER[CH1CFG] = 0b011 EMAC0_MR1[MF] = 0b00 for 10Mbps EMAC0_MR1[MF] = 0b01 for 100Mbps EMAC0_MR1[MF] = 0b10 for 1000Mbps RGMII0_SSR[SP0] ignored Set RGMII0_SSR[SP0] for line speed.
GMII1/MII1 (PPC460GT only)	GMII/MII through the RGMII bridge: EMAC2<—>RGMII Bridge 1<—>GMII1/MII1 SDR0_ETH_CFG[SGMII2_LPBK]=0 SDR0_ETH_CFG[SGMII2_ENABLE] = 0 SDR0_ETH_CFG[GMC0BS] = 0 RGMII1_FER[CH2CFG] = 0b111 RGMII1_FER[CH3CFG] = 0b011 RGMII1_FER[CH2CFG] = 0b111 RGMII1_FER[CH3CFG] = 0b011 EMAC2_MR1[MF] = 0b00 for 10Mbps EMAC2_MR1[MF] = 0b01 for 100Mbps EMAC2_MR1[MF] = 0b10 for 1000Mbps RGMII1_SSR[SP2] ignored Configure GPIO signals.

Table 29-5. RGMII Configuration

RGMII Interface	Configuration
RGMII0	EMAC0<–>RGMII Bridge 0 <–>RGMII0 SDR0_ETH_CFG[SGMII0_LPBK] = 0 SDR0_ETH_CFG[SGMII0_ENABLE] = 0 SDR0_ETH_CFG[GMC0BS] = 0 RGMII0_FER[CH0CFG] = 0b101 RGMII0_FER[CH1CFG] = 0b101 Set EMAC0_MR1[MF] = 0b10 for RGMII. Enables GMII between the EMAC and RGMII bridge. Set RGMII0_SSR[SP0] for line speed..
RGMII1	EMAC1<–>RGMII Bridge 0 <–>RGMII1 SDR0_ETH_CFG[SGMII1_LPBK] = 0 SDR0_ETH_CFG[SGMII1_ENABLE] = 0 SDR0_ETH_CFG[GMC0BS] = 0 RGMII0_FER[CH0CFG] = 0b101 RGMII0_FER[CH1CFG] = 0b101 Set EMAC1_MR1[MF] = 0b10 for RGMII. Enables GMII between the EMAC and RGMII bridge. Set RGMII0_SSR[SP1] for line speed.
RGMII2 (PPC460GT only)	EMAC2<–>RGMII Bridge 1<–>RGMII2 SDR0_ETH_CFG[SGMII2_LPBK] = 0 SDR0_ETH_CFG[SGMII2_ENABLE] = 0 SDR0_ETH_CFG[GMC1BS] = 0 RGMII1_FER[CH2CFG] = 0b101 RGMII1_FER[CH3CFG] = 0b101 Set EMAC2_MR1[MF] = 0b10 for RGMII. Enables GMII between the EMAC and RGMII bridge. Set RGMII1_SSR[SP2] for line speed. Configure the GPIO signals.
RGMII3 (PPC460GT only)	EMAC3<–>RGMII Bridge 1<–>RGMII3 SDR0_ETH_CFG[GMC1BS] = 0 RGMII1_FER[CH2CFG] = 0b101 RGMII1_FER[CH3CFG] = 0b101 Set EMAC1_MR1[MF] = 0b10 for RGMII. Enables GMII between the EMAC and RGMII bridge. Set RGMII1_SSR[SP3] for line speed. Configure the GPIO signals.

Table 29-6. SGMII Configuration

SGMII Interface	Configuration
SGMII0 (PPC460EX / GT only)	EMAC0<-->GPSC0 <-->SGMII SerDes 0 SDR0_ETH_CFG[SGMII0_LPBK] = 0 SDR0_ETH_CFG[SGMII0_ENABLE] = 1 SDR0_ETH0_CFG[SGMII0_SPEED_SEL] = 1 (enables auto-negotiation) **Note:** Auto-negotiation only supported by 460EX/GT rev B. 460EX/GT rev A operates at 1Gps. Set EMAC0_MR1[MF] = 0b11. Enables TBI between the EMAC and internal SGMII SerDes.
SGMII1	EMAC1<-->GPSC1 <-->SGMII SerDes 1 SDR0_ETH_CFG[SGMII1_LPBK] = 0 SDR0_ETH_CFG[SGMII1_ENABLE] = 1 SDR0_ETH0_CFG[SGMII1_SPEED_SEL] = 1 (enables auto-negotiation) **Note:** Auto-negotiation only supported by 460EX/EXr/GT rev B. 460EX/EXr/GT rev A operates at 1Gps. Set EMAC1_MR1[MF] = 0b11. Enables TBI between the EMAC and internal SGMII SerDes.
SGMII2 (PPC460GT only)	EMAC2<-->GPSC2 <-->SGMII SerDes 2 SDR0_ETH_CFG[SGMII2_LPBK] = 0 SDR0_ETH_CFG[SGMII2_ENABLE] = 1 **Note:** SGMII2 does not support auto-negotiation. This port only operates at 1Gbps. Set EMAC2_MR1[MF] = 0b11. Enables TBI between the EMAC and internal SGMII SerDes.

EMAC Gigabit mode Physcial Coding Sublayer (GPSC) registers must be configured when using SGMII mode. The internal communication between the EMAC and SGMII SerDes is ten-bit mode (TBI).

29.1 ZMII Bridge

Media independent interface (MII) signals to and from an EMAC are passed to the ZMII bridge. The ZMII bridge passes MII signals through to the PHY.

29.2 ZMII Features

The ZMII bridge features:

- Support for one MII PHY
- Programmable selection of EMAC0 to drive MII

29.3 ZMII Bridge Interface Signals

Each chip pin (ball) is associated with one or more ZMII bridge signals. The ZMII bridge signal associated with each pin depends on the ZMII bridge interface selection. Please refer to the *Ethernet Interface* section of the *Signal Functional Description* table in the *PPC460EX/EXr/GT Embedded Processor Data Sheet* for a complete list of all of these signals, how they are grouped, and their I/O characteristics.

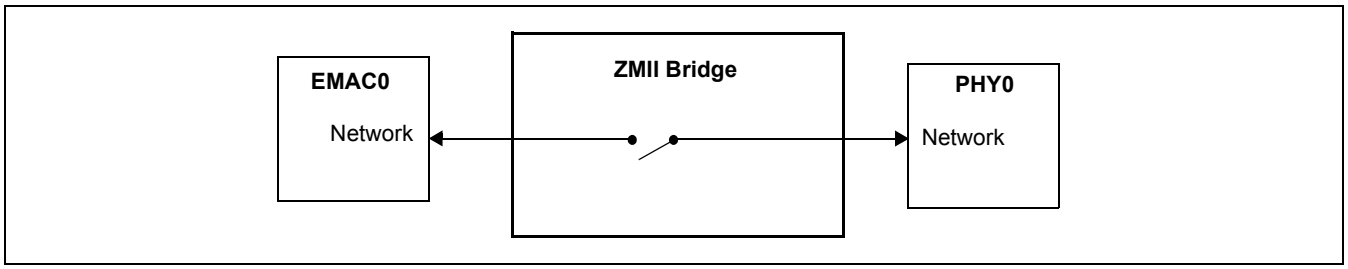
29.4 EMAC-ZMII Bridge Interfaces

The ZMII bridge attaches to the EMACs on the standard MII, which consists of two sections: transmit and receive data. The transmit and receive data sections move data nibbles across the interface at 25MHz and 2.5MHz for 100Mbps or 10Mbps operation, respectively. Control signals define valid data, error conditions, and collision detection.

29.4.1 MII Interface

When the ZMII bridge is configured for MII, the MII signals from one EMAC are simply passed through the ZMII bridge to a PHY. *Figure 29-2* shows an EMAC connected to the ZMII bridge and a PHY. If using MII mode, only a single EMAC may be used.

Figure 29-2. EMAC to PHY Using MII



29.5 ZMII Bridge Registers

This section describes the registers in the ZMII bridge. The ZMII bridge registers are accessed using an OPB slave interface.

Table 29-7. ZMII Registers

Register	Description	Address	Access	Page
ZMII0_FER	ZMII Function Enable Register	0x4 EF60 0D00	R/W	1060

29.5.1 Function Enable Register (ZMII0_FER)

This register enables the ZMII bridge to pass through the MII0 signals from EMAC0..

Figure 29-3. Function Enable Register (ZMII0_FER)			
0:2		Reserved	
3	MII0	EMAC0 MII PHY interface 0 Disable 1 Enable	
4:31		Reserved	

29.6 RGMII Bridge

The RGMII bridge provides the capability for the EMACs to connect to external Ethernet PHYs that support RGMII or standard GMII. A GMII port requires twenty-four pins of data and control, while the RGMII port needs only twelve pins in addition to a common reference clock.

29.6.1 RGMII Bridge Features

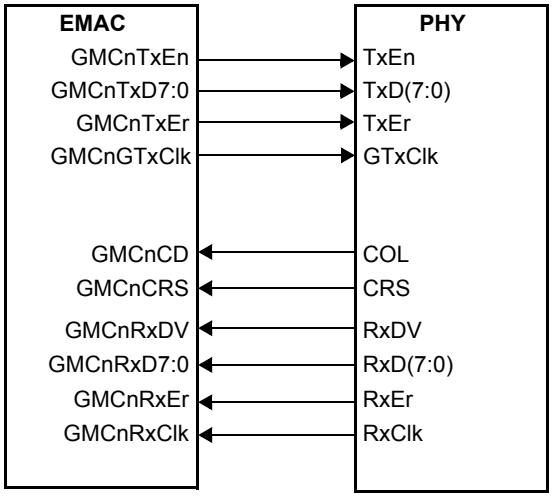
The RGMII bridge supports the following features:
- One GMII PHY
- Two independent RGMII PHYs (PPC460EX/EXr)
- Four independent RGMII PHYs (PPC460GT)
- 10 Mbps, 100 Mbps or 1 Gbps data rates
- Uses single 125 MHz clock reference sourced from an external source
- Independent 4-bit transmit and receive paths
- Complies with 2.5 V CMOS interface voltages as defined by JEDED EIA/JESD8-5

29.6.2 RGMII Bridge Interface

The purpose of the RGMII bridge is to reduce the pin count of the Ethernet gigabit medium independent interface (GMII) by multiplexing the data and control signals.

In the normal GMII mode (as shown in *Figure 29-4*), data between EMAC and PHY is transmitted and received over two eight-bit wide data buses TxD(7:0), RxD(7:0) with six control signals TxEn, TxEr, RxDv, RxEr, COL and CRS. The data and control signals are synchronous to the two clocks GTxClk and RxClk. The total number of pins required by GMII would be 24 plus MDIO and MdClk. (MDIO and MdClk are bidirectional serial data and clock signals used by EMAC to configure the internal registers inside the PHYs.

Figure 29-4. GMII 24 Pins + 2



Note: GMCRefClk is required for GMII modes.

Table 29-8. RGMII Bridge PHY Interface Signals

RGMII	I/O	Description
GMCnGTxClk	I/O	RGMII transmit reference clock is 125MHz, 25MHz, or 2.5MHz ± 50ppm depending on speed
GMCnTxD3:0	O	RGMII transmits bits 3:0 on positive edge of GMCnGTxClk and bits 7:4 on negative edge of GMCnGTxClk
GMCnTxCtl	O	RGMII transmits GMCnTxEn on positive edge of GMCnGTxClk and a logical derivative of GMCnTxEn and GMCnTxEr on negative edge of GMCnGTxClk
GMCnRxClk	O	RGMII continuous receive reference clock is 125MHz, 25MHz, or 2.5MHz ± 50ppm depending on speed
GMCnRxD3:0	I	RGMII receives bits 3:0 on positive edge of GMCnRxClk and bits 7:4 on negative edge of GMCnRxClk
GMCnRxCtl	I	RGMII receives RxEn on positive edge of GMCnRxClk and a logical derivative of GMCnRxEN and GMCnRxER on negative edge of GMCnRxClk
The data rate achievable at each frequency is as follows: • GMCnGTxClk = GMCnRxClk = 125MHz ≥ (125MHz x 2) x 4 bits = 1000Mbps • GMCnGTxClk = GMCnRxClk = 25MHz ≥ (25MHz) x 4 bits = 100Mbps) • GMCnGTxClk = GMCnRxClk = 2.5MHz ≥ (2.5MHz) x 4 bits = 10Mbps		

29.6.3 Ethernet Signal Grouping for Ethernet- to-PHY Bridges

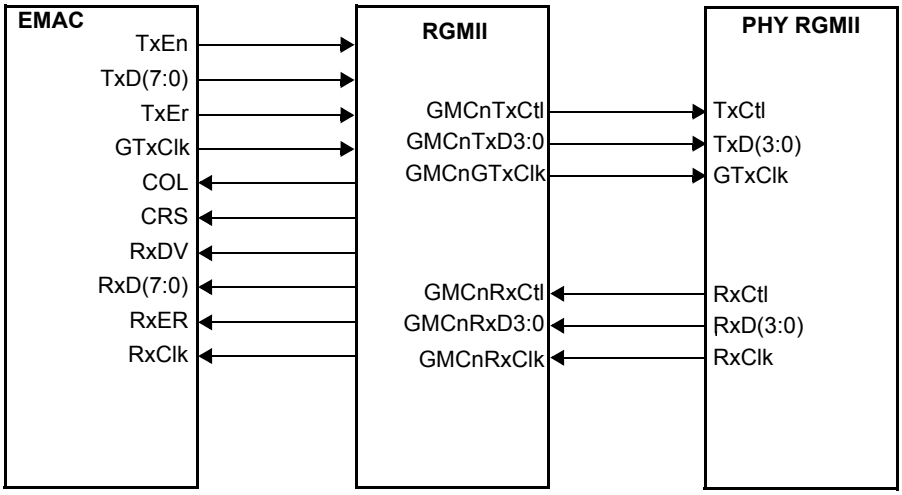
The way a particular signal is used in the Ethernet logic is dependent on the Ethernet mode (MII and GMII) that the logic is supporting. Please refer to the *Ethernet Interface* section of the *Signal Functional Description* table in the *PPC460EX/EXr/GT Embedded Processor Data Sheet* for a complete list of all of these signals, how they are grouped, and their I/O characteristics.

29.6.4 EMAC-RGMII Bridge Interfaces

The GMII interface is multiplexed by the RGMII module, which reduces pins by using both edges of clocks (reducing data buses TxD, RxD by half), multiplexing control pins TxEn/TxEr, RxDv/RxEr on two clock edges, and regenerating indicator signals COL, CRS from data stream bit patterns.

On the receiving path from the PHY going to the EMAC, the RGMII module places the on the RxD(7:0) bus and associated control signals.

Figure 29-5. EMAC to PHY Using RGMII



Note: GMCRefClk is required for GMII modes.

29.6.5 RGMII Bridge Registers

This section describes the registers in the RGMII bridge, which are listed in *Table 29-9*. The RGMII bridge registers are accessed using an OPB slave interface.

Table 29-9. RGMII Bridge Registers

Register	Description	Address	Access	Page
RGMII0_FER	RGMII0 Function Enable Register	0x4 EF60 1500	R/W	1064
RGMII0_SSR	RGMII0 Speed Select Register	0x4 EF60 1504	R/W	1065
RGMII1_FER	RGMII1 Function Enable Register (PPCR460GT only)	0x4 EF60 1600	R/W	1064
RGMII1_SSR	RGMII1 Speed Select Register (PPCR460GT only)	0x4 EF60 1604	R/W	1065

29.6.5.1 RGMII0 Function Enable Register (RGMII0_FER)

The RGMII function enable register (RGMII0_FER) selects the various interface conversion functions provided by the RGMII bridge. RGMII0_FER also deselects (functionally isolates) an unused EMAC. GMII and RGMII modes are mutually exclusive. However, both EMAC interfaces can be used simultaneously in RGMII mode.

Figure 29-6. RGMII0 Function Enable Register (RGMII0_FER)

0:24		Reserved	
25:27	CH1CFG	EMAC1 Configuration 0xx Disable 100 Reserved 101 RGMII1 110 Reserved 111 Reserved	
28		Reserved	
29:31	CH0CFG	EMAC0 Configuration 0xx Disable 100 Reserved 101 RGMII0 110 Reserved 111 MII0/GMII0 (passthrough)	

Note: In reduced mode, the two EMACs operate independently. In pass through mode, only one EMAC operates. If both EMACs are enabled and set to pass-through, or one EMAC is reduced and the other is pass-through, these are invalid settings resulting in both EMACs being disabled.

29.6.5.2 RGMII1 Function Enable Register (RGMII1_FER) (PPC460GT only)

Figure 29-7. RGMII1 Function Enable Register (RGMII1_FER)

0:24		Reserved	
25:27	CH3CFG	EMAC3 Configuration 0xx Disable 100 Reserved 101 RGMII3 110 Reserved 111 Reserved	
28		Reserved	
29:31	CH2CFG	EMAC2 Configuration 0xx Disable 100 Reserved 101 RGMII2 110 Reserved 111 MII1/GMII1 (passthrough)	

29.6.5.3 Speed Selection Register (RGMII0_SSR)

The speed selection register selects the speed at which each EMAC transfers data (10Mbps, 100Mbps, or 1000Mbps). When 10Mbps is selected, the RGMII bridge generates a 2.5MHz clock. When 100Mbps is selected, the RGMII bridge generates a 25MHz clock. When 1000Mbps is selected, the RGMII bridge generates a 125MHz clock. This register also controls whether collisions are indicated.

Figure 29-8. Speed Selection Register (RGMII0_SSR)

0:20		Reserved	
21:23	SP1	EMAC 1 Speed Selection 000 10 Mbps selected. 010 100 Mbps selected. 100 1000 Mbps selected	
24:28		Reserved	
29:31	SP0	EMAC 0 Speed Selection 000 10 Mbps selected. 010 100 Mbps selected. 100 1000 Mbps selected	

29.6.5.4 Speed Selection Register (RGMII1_SSR)

Figure 29-9. Speed Selection Register (RGMII1_SSR)

0:20		Reserved	
21:23	SP3	EMAC 3 Speed Selection 000 10 Mbps selected. 010 100 Mbps selected. 100 1000 Mbps selected	
24:28		Reserved	
29:31	SP2	EMAC 2 Speed Selection 000 10 Mbps selected. 010 100 Mbps selected. 100 1000 Mbps selected	

29.7 SGMII Bridge

The SGMII bridge provides the capability for the EMACs to connect to external Ethernet PHYs that support SGMII. A SGMII port requires six pins of differential data and clocks.

29.7.1 SGMII Bridge Features

The SGMII bridge supports the following features:
- Two independent SGMII PHYs (PPC460EX)
- One independent SGMII PHY (PPC460EXr)
- Three independent SGMII PHYs (PPC460GT)
- 10Mbps, 100Mbps or 1Gbps data rates
- Uses single 1250MHz clock reference sourced from Ethernet PLL
- Independent serial transmit and receive paths

29.7.2 SGMII Bridge Interface

The SGMII bridge minimizes the pin count of the Ethernet gigabit medium independent interface (GMII) by serializing the data and control signals.

29.7.3 SGMII Bridge Registers

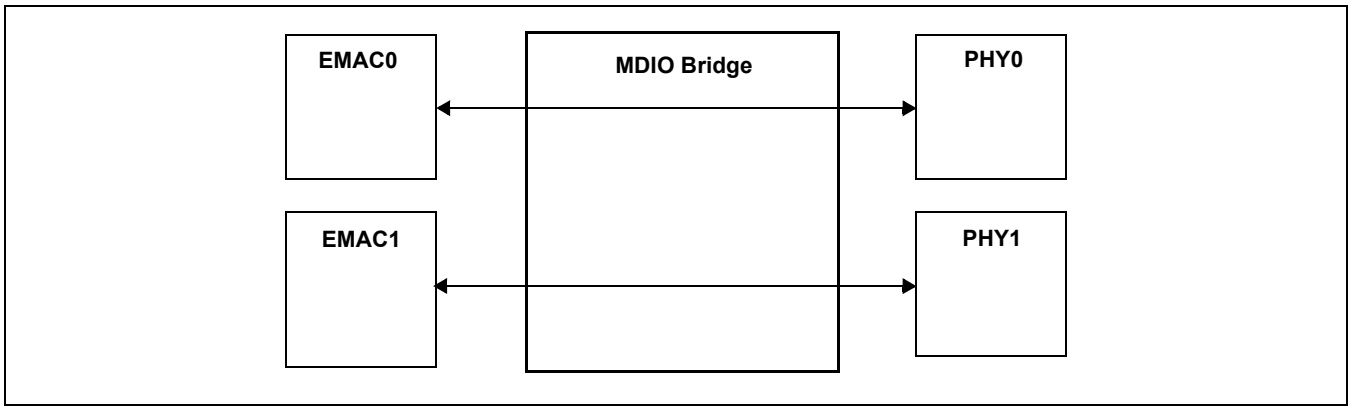
SDR register fields related to the SGMII bridge can be found in the Ethernet Configuration Register (SDR0_ETH_CFG) and Ethernet Status Register (SDR0_ETH_STS).

Specifically, the SGMIIx_ENABLE field in the SDR0_ETH_CFG register controls enabling/disabling the port. This register also has some loopback controls. Loss of signal and clock status can be observed in the SDR0_ETH_STS register.

29.8 MDIO Bridge

The MDIO Bridge interface allows software to select which EMAC controls access to the registers in the PHY. It consists of a single bidirectional data signal and a clock driven from the MDIO Bridge.

Figure 29-10. EMAC to PHY Using MDIO



Ethernet Configuration Register (SDR0_ETH_CFG) describes the SDR registers related to the MDIO bridge. Specifically, the MDIO_SEL field selects which EMAC controls the data and clock signals.

30. TCP/IP Accelerator Hardware (TAH)

The PPC460EX/EXr/GT provides two TCP/IP acceleration hardware controllers (TAHs) that are inserted between the memory access layer (MAL) and the two gigabit Ethernets (EMAC0 and EMAC1). Both TAHs are implemented identically with the exception of register addresses. Because both TAHs operate identically, this chapter uses the word TAH to refer to TAH0 and TAH1. Please note that the hardware acceleration functions apply to both EMACs. Hence the word EMAC in this chapter denotes EMAC0 and EMAC1.

30.1 Overview

The TAH provides hardware acceleration functions for the EMAC (which support 10/100/1000 Mbps operations) to improve bandwidth and lower processor utilization. It provides checksum verification for TCP/UDP/IP headers in the receive path, checksum generation for TCP/UDP/IP headers in the transmit path, and TCP segmentation support in the transmit path. To utilize the acceleration function in the TAH, the IP datagram must use the ethernet encapsulation (RFC894) or IEEE802.2/802.3 encapsulation (RFC1042). The TAH can handle VLAN-tagged frames and support standard and jumbo packets.

The MAL is an intermediate processing layer which manages data transfers between the TAH (or the EMAC if no TAH is present) and memory. The MAL performs functions such as arbitration between service requests, handling the buffer descriptor memory structure, and updating the descriptor status/control field at the end of packet transfer. A software device, such as the TCP/IP protocol stack, uses the buffer descriptor to inform the MAL about buffer locations and packet or buffer status. The MAL uses the buffer descriptors to convey packet transfer status from the EMAC back to the protocol stack. For more information see *Memory Access Layer* on page 949.

For receive packets, setting the Checksum Verification on Receive (CVR) bit in the accelerate mode register, TAHx_MR, enables hardware checksum verification for all incoming packets for a given EMAC/TAH.

For transmit packets, hardware generated checksums and/or packet segmentation is done on a per-packet basis. This is accomplished by setting the proper bits in the descriptor control/status field of the buffer descriptor. The size that packets should be segmented into (if segmentation is enabled) is controlled by one of six Segment Size Registers (TAHx_SSRx). Bits in the buffer descriptor also determine which SSR is used.

EMAC is a generic implementation of the Ethernet Media Access Control (EMAC) protocol compliant with ANSI/IEEE Std 802.3 and IEEE 802.3u supplement. The EMAC supports both Half Duplex (CSMA/CD) and Full Duplex operation. The EMAC communicates with the physical (PHY) device using a Gigabit Media Independent Interface (GMII). The EMAC also includes a Gigabit Ethernet PHY Coding Sublayer (GPCS4). The use of the GPCS4 is optional, and it allows the EMAC to be connected to the PMA interface, and thus to a Gigabit Ethernet layer directly. For more information see *Ethernet Media Access Controller* on page 995.

The TAH interfaces to the MAL as an Extended OPB (EOPB) slave on behalf of the EMAC, and interfaces to the EMAC as an Extended OPB (EOPB) master on behalf of the MAL. On each EOPB interfaces, a separate receive and transmit channel is supported. It also provides an on-chip peripheral bus (32-bit bidirectional OPB) slave interface for access to the configuration and status registers. The TAH maintains a transmit FIFO to support the hardware checksum function on the transmit side.

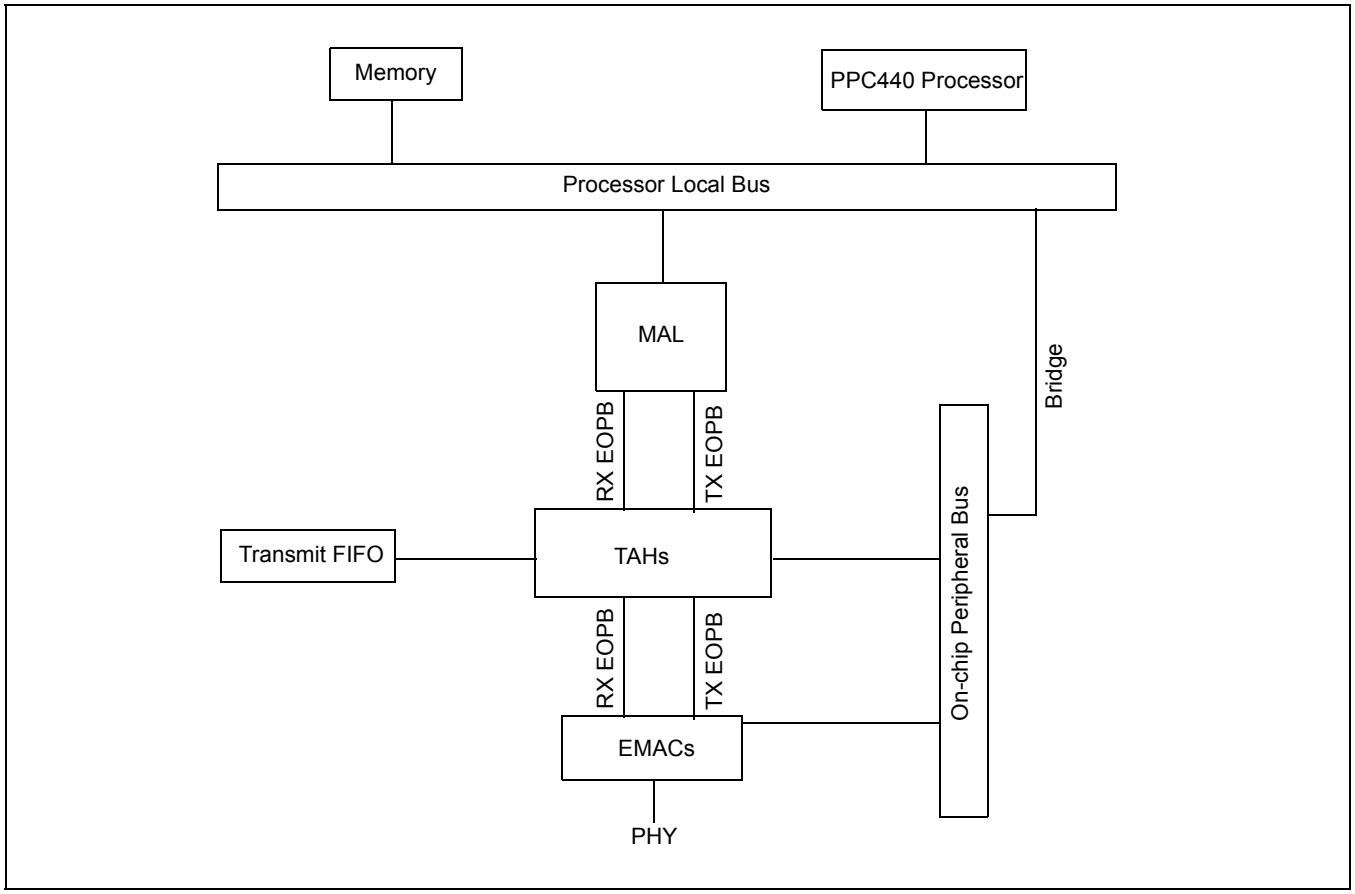
Network bandwidth has increased at a much faster pace than processor and memory performance. Because of these limitations, CPU utilization can easily reach 100% long before the network is saturated. Although 1Gb Ethernet is ten times faster than 100Mb, the overhead of packet processing in software may result in only a 3x or 4x increase in overall throughput. By off-loading some of the packet processing that is traditionally done by software, significant performance increases can be realized. The TCP Acceleration Hardware provides this capability in the PPC460EX/EXr/GT.

30.1.1 Features

- Verifies checksum for the UDP/TCP/IP headers in the receive path
- Generates checksum for the UDP/TCP/IP headers in the transmit path
- Supports TCP segmentation in the transmit path
- Supports IPv4, IPv6, and IPv6 within IPv4 (receive path only) headers
- Supports ethernet encapsulation (RFC894)
- Supports IEEE 802.2/802.3 encapsulation (RFC1042)
- Supports VLAN tagged frames according to IEEE 802.3ac
- Inserts/Replaces VLAN Tag for transmit packets (programmable option)
- Inserts/Replaces MAC source address for transmit packets (programmable option)
- Inserts Frame Check Sequence (FCS) control for the transmitted/received packets
- Provides OPB interface for access to internal registers
- Provides two 128-bit MAL EOPB interfaces (separate transmit and receive channels) with the MAL for moving packets (operates at PLB frequency)
- Provides two 128-bit MAL EOPB interfaces (separate transmit and receive channels) with the EMAC for moving packets (operates at PLB frequency)
- Provides 10KB of transmit FIFOs
- Supports multiple status mechanism (called "back to back TX status mechanism") which allows the TAH to delay the sending of the current status word or to piggyback the previous status word with the current status word

Figure 30-1 illustrates a general system structure overview of an embedded PowerPC processor integrated with TCP/IP acceleration hardware on Ethernet.

Figure 30-1. General PPC460EX/EXr/GT Structure with TCP/IP Acceleration Hardware



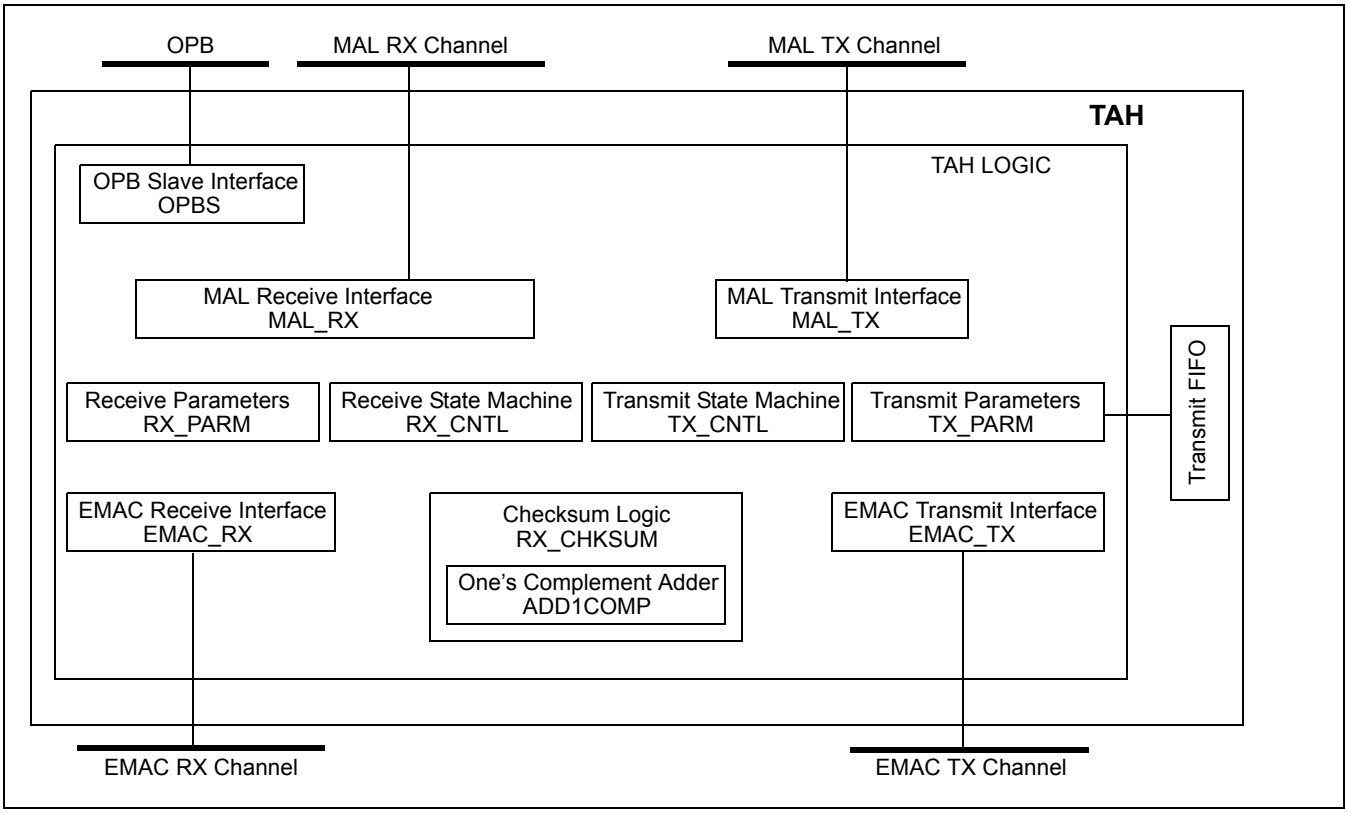
The TAH may be connected to a single EOPB bus. On this bus, both the OPB master and the MAL are connected as masters. The OPB master will use only the OPB function while the MAL will use the EOPB over this single EOPB bus.

The TAH's transmit channel and receive channel operate independently. This means that each of these channels can be connected to a different MAL instance, possibly on a different EOPB (this option can increase system performance especially when Gigabit mode is chosen).

30.2 TAH Operations

The TAH hardware components and its internal structure are illustrated in *Figure 30-2* followed by a brief description of its components.

Figure 30-2. Internal TAH Structure



30.2.1 TAH Hardware Components

TAH hardware components consist of:

- EOPB slave device to the MAL
- EOPB master device to the EMAC
- Hardware acceleration logic on the transmit path
- Hardware acceleration logic on the receive path
- OPB slave device

30.2.1.1 EOPB Slave Logic to the MAL

The MAL slave logic consists of two finite state machines. MAL_TX is the finite state machine for the transmit path and MAL_RX is the finite state machine for the receive path. The TAH is not a generic EOPB slave, but rather a dedicated MAL slave.

30.2.1.2 EOPB Master Logic to the EMAC

As a surrogate for the MAL, the TAH provides the EOPB master logic in two finite state machines. EMAC_TX is the finite state machine for the transmit path and EMAC_RX is the finite state machine for the receive path. The TAH is not a generic EOPB master, but rather a dedicated EMAC master.

30.2.1.3 Hardware Acceleration Logic on the Transmit Path

This is implemented as a finite state machine (TX_CNTL) and associated parameters (TX_PARM). These modules also provide the control signals for the external Transmit FIFO. The checksum logic is contained in ADD1COMP and RX_CHKSUM.

30.2.1.4 Hardware Acceleration Logic on the Receive Path

This is implemented as a finite state machine (RX_CNTL) and associated parameters (RX_PARM). The checksum logic is contained in ADD1COMP and RX_CHKSUM.

30.2.1.5 OPB Slave (OPBS) Logic

The OPB slave (OPBS) logic handles the OPB transactions for accessing the TAH's configuration and status registers. It provides two-cycle latency for single data transfer. It does not support burst transfers or the OPB Sequential Address mechanism.

30.2.2 Data Ordering—Transmit and Receive Data Path

The TAH performs data transfers between the MAL EOPB and the EMAC EOPB. The EOPB operates using Big Endian format in quadword mode (128 bits). The TAH preserves the data ordering between the two interfaces.

30.2.3 TAH Transmit Operation

The transmit part of the TAH is responsible for frame transmission from MAL to the EMAC. The EMAC's TX channel can be configured to work in either of the following two ways:

1. Single packet mode: the Transmit Request bit in Mode Register 1 is 0. The channel requests a single packet from the TAH and then resets its GET_NEW_PACKET bit in the Transmit Mode Register 0 when it receives a TAH2EMAC_TX_STATUS_DONE indication. The channel will only ask for service after the GET_NEW_PACKET bit is set to 1 again.
2. Multiple packet mode: the Transmit Request bit in Mode Register 1 is 1. Once the channel finishes transferring a packet and receives TAH2EMAC_TX_STATUS_DONE, it will ask the TAH for the next packet if there is enough room in the FIFO. The channel will keep asking for more packets until either of the following events occur:
    - The channel receives TAH2EMAC_TX_DSCR_NOT_VALID at which time it will clear its GET_NEW_PACKET bit and wait for software to re-activate it by writing 1 to GET_NEW_PACKET bit.
    - A Transmit Error or an SQE occurs and the corresponding interrupt is not masked in the Interrupt Status Enable Register. After such an error, the channel clears its GET_NEW_PACKET bit and activates its DB bit and the corresponding ERROR bit in the Interrupt Status Register, as soon as it receives TAH2EMAC_TX_STATUS_DONE indication. The channel will not request service again until the GET_NEW_PACKET bit is set and the DB bit is cleared.

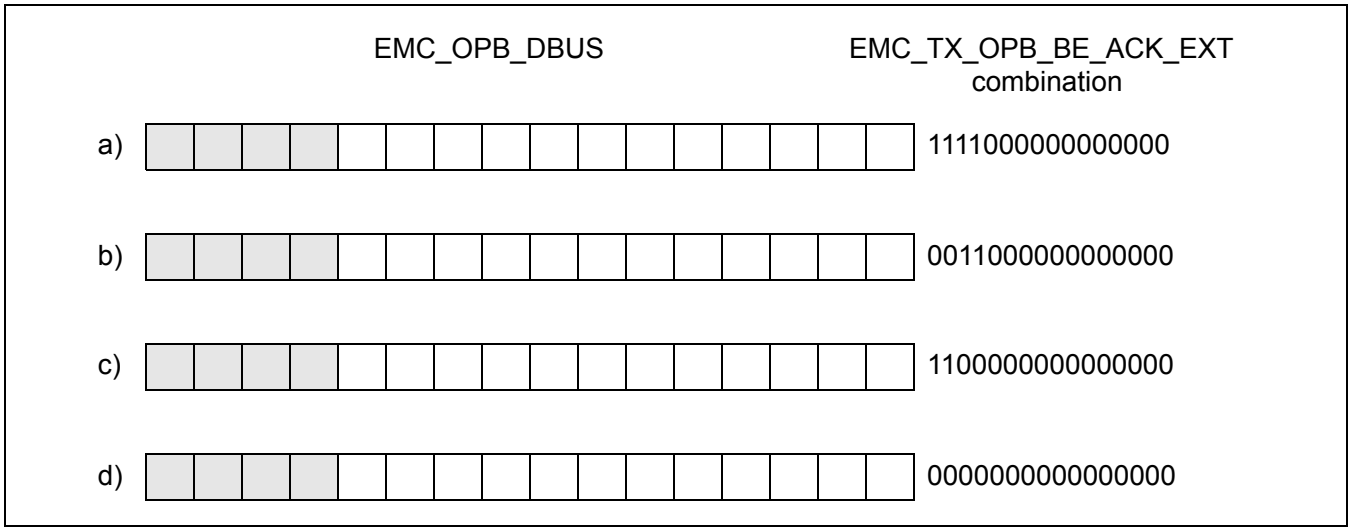
There are two situations in which the EMAC TX channel may request data transfer from the TAH.

1. If the channel is in the Idle Phase (after reset or after the channel received TAH2EMAC_TX_STATUS_DONE), The EMAC may assert EMC_TX_FRAME and drive EMC_TX_ARB_LEVEL to a non-idle value.
2. If the channel requested data transfer and received TAH2EMAC_TX_DSCR_NOT_VALID, then EMC_TX_FRAME is already asserted, so when the channel is ready to receive data again, it only needs to drive EMC_TX_ARB_LEVEL to a non-idle value.

At the end of the transmission process, the EMAC can be configured to either wait for every status/word before requesting a new packet, or not waiting for status/error word after packet transmission:

1. Waiting for every status/error word before requesting a new packet: The Maximum Waiting Status Words bits in Mode Register 1 are `000'. The channel waits until it gets the status word after EMC_TX_FRAME deassertion, and only after the status word is ready EMC_TX_ARB_LEVEL is driven to a non-idle value, EMC_OPB_XFER_ACK is pulsed, and EMC_TX_OPB_BE_ACK_EXT is 1100000000000000 (see *Figure 30-3*, case c).
2. Not Waiting for status/error word after packet transmission: The Maximum Waiting Status Words bits in Mode Register 1 are 001. After EMC_TX_FRAME deassertion, EMC_TX_ARB_LEVEL is driven to a non-idle value, EMC_OPB_XFER_ACK is pulsed, and the EMC_TX_OPB_DBUS can only be assembled in one of the following ways as described in *Figure 30-3*:
3.

Figure 30-3. EMC_OPB_DBUS Assembly



Note: Shaded boxes in *Figure 30-3* indicate the location where the valid status/error words should be placed on the EMC_OPB_DBUS during the status/error/transfer. Cases a, b, c, and d are described below.

a) Two valid status/error words - The status/error word of the current transmit packet is placed on the first and second bytes, and the status/error word of the previous transmit packet is placed on the third and fourth bytes.

b) One valid status/error word of the previous transmit packet - The status/error word is placed on the third and fourth bytes. (The status of the current packet should be ready after the next transmit packet).

c) One valid status/error word of the current transmit packet - The status/error word is placed on the first and second bytes. (The status of the previous packet was sent).

d) Null status/error word—no status/error word is sent. (The status of the current packet should be ready after the next transmit packet and the previous packet was sent).

The interface between the TAH and the MAL is similar to the interface between the TAH and the EMAC as described above. The only difference is in the handling of the status/error word when hardware segmentation is enabled. For more details, see the following section on hardware acceleration support.

30.2.3.1 Hardware Acceleration Support

The TAH provides hardware acceleration in the transmit path in the form of hardware checksum generation and TCP segmentation support. Since the TAH will modify the packet content, the application software must enable the option in the control information to perform data padding and to add FCS.

When hardware acceleration is enabled, certain restrictions are imposed on the TCP/UDP/IP headers. The restrictions on the IP header fields are listed in *Table 30-1*:

Table 30-1. IPv4 Header Fields in Support of Hardware Acceleration Functions

	Verify Checksum on Receive	Generate Checksum Only on Transmit	TCP Segmentation
Version	Must be IPv4	Must be IPv4	Must be IPv4
Header Length	Must be 5 (no options allowed)	Must be 5 (no options allowed)	Must be 5 (no options allowed)
Type of Service	Don't care	Don't care	Don't care
Total Length	Don't care	Must be more than 40 See **Note**.	Initial value must be more than 40. Actual value used is generated by hardware based on selected segment size.
ID	Don't care	Don't care	Generated by hardware based on initial value
Flag	Fragment bit must be 0	Fragment bit must be 0	Fragment bit must be 0
Fragment Offset	Must be 0	Must be 0	Must be 0
Time-to-Live	Don't care	Don't care	Don't care
Protocol	Must be TCP or UDP or IPv6 within IPv4(0x29)	Must be TCP or UDP	Must be TCP
Header Checksum	Don't care	Generated by hardware	Generated by hardware
Source IP Address	Don't care	Don't care	Don't care
Destination IP Address	Don't care	Don't care	Don't care
Options	Don't care	Don't care	Ignored
Note: The TAH uses the total length field information in the IP header to determine the number of bytes to be processed. It is very important that the number of bytes in the buffer that are to be transmitted matches exactly the total length in the IP header. If the number of bytes are smaller, an error will be reported and the transmission terminated as soon as the error is detected. (Note that when TCP segmentation is enabled, some segments might have already been sent.) If the number of bytes are larger, the padding bytes must be zeros. Non-zero padding bytes can result in checksum errors detected at the receiving site.			

The restrictions on the IPv6 header fields are listed in *Table 30-2*:

Table 30-2. IPv6 Header Fields in Support of Hardware Acceleration Functions

	Verify Checksum on Receive	Generate Checksum Only on Transmit	TCP Segmentation
Version	Must be IPv6	Must be IPv6	Must be IPv6
Traffic Class	Don't care	Don't care	Don't care
Flow Label	Don't care	Don't care	Don't care
Payload Length	Don't care	Must be more than 20 (for TCP) or 8 (for UDP) See **Note**.	Must be more than 20 (for TCP) or 8 (for UDP). Actual value used is generated by hardware based on selected segment size.
Next Header	Must be TCP or UDP	Must be TCP or UDP	Must be TCP
Hop limit	Don't care	Don't care	Don't care
Source IP Address	Don't care	Don't care	Don't care
Destination IP Address	Don't care	Don't care	Don't care
Options	Don't care	Don't care	Ignored
Note: The TAH uses the total length field information in the IP header to determine the number of bytes to be processed. It is very important that the number of bytes in the buffer that are to be transmitted matches exactly the total length in the IP header + 40bytes (IPv6 main header length). If the number of bytes are smaller, an error will be reported and the transmission terminated as soon as the error is detected. (Note that when TCP segmentation is enabled, some segments might have already been sent.) If the number of bytes are larger, the padding bytes must be zeros. Non-zero padding bytes can result in checksum errors detected at the receiving site.			

The restrictions on the TCP header fields are listed in *Table 30-3*:

Table 30-3. TCP Header Fields in Support of Hardware Acceleration Functions

	Verify Checksum on Receive	Generate Checksum Only on Transmit	TCP Segmentation
Source Port Number	Don't care	Don't care	Don't care
Destination Port Number	Don't care	Don't care	Don't care
Sequence Number	Don't care	Don't care	Generated by hardware based on initial value
Acknowledge Number	Don't care	Don't care	Don't care
Header Length	Don't care	Don't care	First packet uses value provided; subsequent packets use a value of 5
Flags	Don't care	Don't care	See **Note**
WIndow Size	Don't care	Don't care	Don't care
TCP Checksum	Don't care	Generated by hardware	Generated by hardware

Table 30-3. TCP Header Fields in Support of Hardware Acceleration Functions (Continued)

	Verify Checksum on Receive	Generate Checksum Only on Transmit	TCP Segmentation
Urgent Pointer	Don't care	Don't care	Don't care
Options	Don't care	Don't care	Don't care

Note: When TCP segmentation is enabled, the TCP flags are handled as follows:
- URG (urgent)—recreated in the first segment only; reset in all other segments
- ACK (acknowledge)—recreated in all segments
- PSH (push)—recreated in the last segment only; reset in all other segments
- RST (reset)—must be zero
- SYN (synchronize)—must be zero
- FIN (finished)—recreated in the last segment only; reset in all other segments.

30.2.3.2 Hardware Checksum Generation

The TAH generates the IP, UDP, and TCP checksums for the packet when the appropriate bits in the control information (see the section on MAL TX Control) from the MAL are set to enable this feature.

When the EMAC requests data transfer from the TAH, the TAH will in turn request data transfer from the MAL. If hardware checksum generation is enabled in the control information returned from the MAL, then the TAH will store the subsequent data temporarily in the Transmit FIFO while it calculates the checksum. Data is not sent to the EMAC until the entire packet has been transferred from the MAL to the TAH since the checksum must be computed over the entire packet. When data is finally sent to the EMAC, the computed checksums will be placed in the appropriate header fields in the packet. The TAH parses the incoming data stream to determine the type of data present. The following formats are supported:

- IP version 4 and version 6
- TCP or UDP
- Ethernet II (DIX) frame format as defined in RFC 894 or IEEE 802 frame format with SNAP header as defined in RFC 1042
- VLAN as defined in IEEE 802.3ac

30.2.3.3 Hardware Segmentation

The TAH segments the TCP packet from the MAL into smaller packets using the selected Segmentation Size when the appropriate bits in the control information (see the section on MAL TX Control) from the MAL are set to enable this feature. By enabling the segmentation support in hardware, the application software can transfer a large block of data, say 32K bytes, and provide a single TCP/IP header template, and have hardware segment the data into multiple packets of the desired segment size, say 1.5K bytes, and insert the necessary TCP/IP header for each packet. The desired segment size is specified in *TAH Segment Size Registers 0:5 (TAHx_SSR0–TAHx_SSR5)* on page 1085.

When the EMAC requests data transfer from the TAH, the TAH will in turn request data transfer from the MAL. If hardware segmentation is enabled in the control information returned from the MAL, then the TAH will store the subsequent data temporarily in the Transmit FIFO while it calculates the checksum. This is similar to the procedure for supporting hardware checksum generation. The difference is in the handling of the TCP/IP headers. When segmentation is enabled, the original headers from the packet received from the MAL is saved for later use.

New headers computed based on the original headers are stored in the Transmit FIFO instead. The header fields affected are described in *Hardware Acceleration Support* on page 1073. One of the fields affected is the IP Length field. This is computed from the selected Segment Size. When the amount of data equal to the selected segment size has been transferred from the MAL, the TAH will pause the transfer to store the computed checksum in the Transmit FIFO at the location where the headers are stored. The TAH will also start transferring the segmented packet from the Transmit FIFO to the EMAC. At the same time, the TAH will use the stored headers to create the

headers for the next packet and continue the process of transferring data from the MAL. This process continues until all the data from the MAL has been transferred.

As the smaller packets are sent to the EMAC, status will be returned for each packet. If error is reported by the EMAC, then the transmit operation is terminated and the error is reported to the MAL. If no error is reported and there are more segments to be sent for the original packet from the MAL, then the TAH will not forward the status to the MAL since the MAL only expects a single status for the entire packet. At the end of the segmentation operation, after the entire packet from the MAL has been segmented into smaller packets and sent to EMAC, the final status from EMAC is reported to the MAL. For optimum performance, the application software should set the Maximum Waiting Status Words bits in the EMAC Mode Register 1 to 0b001 (see *Mode Register 1 (EMACx_MR1)* on page 1022) to enable Back to Back TX Status Mechanism. This allows the status reporting by GMAC to be overlapped with the data transfer between the EMAC and the TAH.

The TAH provides hardware segmentation support for both Ethernet II format frames and IEEE SNAP format frames. In addition, when Ethernet II format frames are used, the TAH also provides hardware segmentation support for Jumbo frames. Note that hardware segmentation for Jumbo frames in the IEEE SNAP format is not supported since there is no established standard.

When hardware segmentation is used for IEEE SNAP format frames, software MUST provide the correct MAC length in the original packet header based on the size of the first segment. Thereafter, hardware will fill in the correct MAC length for subsequent TCP segments.

30.2.3.4 MAL Transmit Data

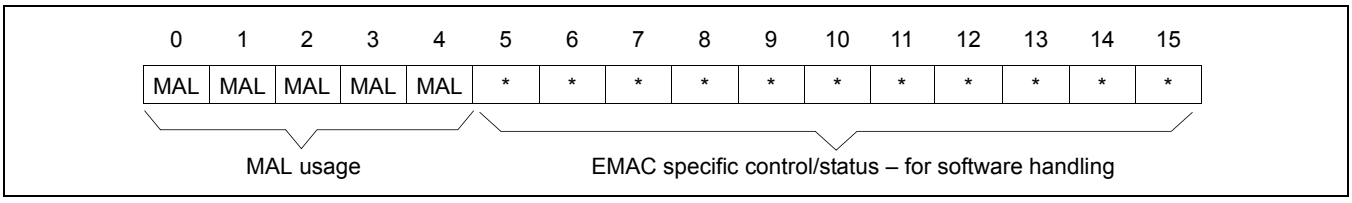
During transmit operations, the low order 3 bits of the MAL address bus (MAL_OPB_ABUS) define the format of the data being transferred from the MAL to the TAH (see *TX Buffer Descriptor Status Control Fields* on page 962). The the MAL data bus (MAL_OPB_DBUS) contains status from TAH/EMAC for previously processed packets when the low order 3 bits of the MAL address bus are all zeros during the Status phase.

MAL TX Write Data - The MAL data bus (MAL_OPB_DBUS) contains packet data to be transmitted when the low order 3 bits of the MAL address bus are non-zero. The MAL data bus (MAL_OPB_DBUS) contains packet related control information from the MAL when the low order 3 bits of the MAL address bus are all zeros during the Frame phase.

30.2.3.5 MAL TX Descriptor Control/Status Field

For each transmitted packet, the MAL uses the descriptor control/status field of the buffer descriptor to provide an EMAC with control information (write), and to obtain packet status from the EMAC after transmission is complete (read). Software writes the control bits in the buffer descriptor before packet transmission, and reads the status bits from the buffer descriptor after packet transmission has completed. See *Buffer Descriptor (BD) Structure* on page 954 for more information on the buffer descriptor structure.

Figure 30-4. MAL TX Descriptor Control/Status Field



Bits	Bit Name	Bit Description	Mode
0:4	Reserved	Always zero	
TX Control Information (Write Access)			
5	Reserved	Always zero	W
6	Generate FCS	0 FCS is not generated by the EMAC. 1 The EMAC calculates and adds the FCS field to the packet to be transmitted. This bit must be set to 1 when hardware acceleration bits 12:14 are enabled.	W
7	Generate padding	0 Padding is not generated by the EMAC. 1 The EMAC adds the padding field to the packet to be transmitted (only when Generate FCS is also set). This bit must be set to 1 when hardware acceleration bits 12:14 are enabled.	W
8	Insert source address	0 The EMAC will not insert source address. 1 The EMAC inserts the source address field into the packet to be transmitted using the stored value.	W
9	Replace source address	0 The EMAC will not replace source address. 1 The EMAC replaces the source address field in the packet to be transmitted using the stored value.	W
10	Insert VLAN Tag	0 The EMAC will not insert a VLAN tag. 1 The EMAC inserts the VLAN Tag field into the packet to be transmitted using the stored value.	W
11	Replace VLAN Tag	0 The EMAC will not replace the VLAN tag. 1 The EMAC replaces the VLAN Tag field in the packet to be transmitted using the stored value.	W
12:14	Hardware acceleration	000 Hardware acceleration is disabled 001 Hardware checksum generation enabled TCP segmentation enabled using TAHx_SSR0 010 Hardware checksum generation enabled TCP segmentation enabled using TAHx_SSR1 011 Hardware checksum generation enabled TCP segmentation enabled using TAHx_SSR2 100 Hardware checksum generation enabled TCP segmentation enabled using TAHx_SSR3 101 Hardware checksum generation enabled TCP segmentation enabled using TAHx_SSR4 110 Hardware checksum generation enabled TCP segmentation enabled using TAHx_SSR5 111 Hardware checksum generation enabled	W
15	Reserved	Always zero	W
TX Status Information (Read Access)			
5	Parity Error	0 No parity error occurred. 1 Indicates parity error occurred.This bit is set when TCP segmentation is enabled.	R
6	Bad FCS on transmitted frame	0 Packet transmission OK. 1 Indicates that bad FCS was indicated for transmitted package.This bit is set when TCP segmentation is enabled.	R
7	Transmit Error Detected	0 Packet transmission OK. 1 Indicates that hardware acceleration is enabled but an error has been detected. TAHx_TSR register specifies the source of error.	R
8	Loss of carrier sense	0 No loss of carrier. 1 During the transmission of a frame, the PHY_CRS input was deasserted after it previously was asserted, or it was not asserted at all. This bit is set when TCP segmentation is enabled.	R

Bits	Bit Name	Bit Description	Mode
9	Excessive deferral	0 No excessive deferral. 1 Indicates that the current frame has been deferred for an excessive period of time. Applicable only in half duplex mode. The value of this period in bit times is calculated in the following ways: For 10/100 Mbps operation it is: 2 x (maxFrameSize x 8) bit times. For 1000 Mbps operation it is: 2 x (burstlimit + maxFrameSize x 8 + headerSize) bit times. This bit is set when TCP segmentation is enabled.	R
10	Excessive collisions	0 Less than 16 collisions. 1 Indicates that the current frame transmission had ended with a collision on the 16th consecutive attempt. Applicable only in half-duplex mode. Returns '0' in full-duplex mode. This bit is set when TCP segmentation is enabled.	R
11	Late collision	0 No late collision. 1 Frame collided outside of the collision window. Applicable only in half-duplex mode. Returns '0' in full-duplex mode. This bit is set when TCP segmentation is enabled.	R
12	Multiple collision	0 More than 1 but less than 16 collisions did not occur. 1 Transmitted frame collided more than once but less than 16 times. Applicable only in half-duplex mode. Does not result in any loss of data and will not be reported when TCP segmentation is enabled.	R
13	Single collision	0 Single collision did not occur. 1 Activates if transmitted frame collided once. Applicable only in half-duplex mode. Does not result in any loss of data and will not be reported when TCP segmentation is enabled.	R
14	Underrun	0 Underrun did not occur. 1 Frame transmission was aborted because of underrun; data from the Transmit FIFO was not valid in time to allow continuous data transmission on the MII/ GMII interface. This bit is set when TCP segmentation is enabled.	R
15	SQE	0 Signal Quality Error did not occur. 1 Signal Quality Error test failed during packet transmission. Applicable only in half -duplex mode during 10 Mbps operation. This bit is 0 in all other modes. This bit is set when TCP segmentation is enabled.	R

The TAH does not monitor MAL_TX_OPB_QW_XFER output from the MAL because all MAL transfers are quadword EOPB transactions.

The EMAC controls the pace of the data prefetch rate by managing the content of the EMC_TX_ARB_LEVEL bus. When hardware acceleration is not enabled, the TAH replicates the EMC_TX_ARB_LEVEL signal from the TAH/ EMAC EOPB to the TAH/MAL EOPB. When hardware acceleration is enabled, since the TAH stores the data in a Transmit FIFO until the entire packet (or segments thereof) has been transferred, the TAH only uses the "normal" setting on the TAH2MAL_TX_ARB_LEVEL bus.

If the EMAC transmit channel is idle, the EMAC may request service at any time by asserting EMC_TX_FRAME. It may withdraw its request for service only when TAH2EMAC_TX_ACTIVE is asserted. Upon detecting an active request from the EMAC in the idle state, the TAH will in turn assert TAH2MAL_TX_FRAME to request service from the MAL. In the middle of a TCP segmentation operation, the TAH will not be in the idle state when the EMAC asserts the service request from its idle state. This is because the TAH has already received part or all of the packet from the MAL and is in the process of breaking the packet into smaller segments for transfer to the EMAC. In this case the TAH will regenerate the control information sent earlier from the MAL, and send the next segmented packet when it is ready.

30.2.3.6 Normal Packet Termination

The last data transfer cycle of the given packet can be determined by examining the least three significant bits of OPB_ABUS. The number of valid bytes within the last data cycle is specified by the content of the MAL_BE_EXT bus.

30.2.3.7 Early Packet Termination in Transmit

The EMAC can initiate an early packet termination during a transmit operation before the TAH completes the data transfer by deasserting EMC_TX_FRAME. This is typically used when error conditions force the EMAC to abort the transmission. Likewise, the TAH can initiate an early packet termination before the MAL completes the data transfer by deasserting TAH2MAL_TX_FRAME if the EMAC initiates early packet termination.

During TCP segmentation operation, the TAH will initiate early packet termination on the MAL interface if the EMAC reports any of the following errors:

- Parity Error
- Bad FCS
- Loss of carrier sense
- Excessive Deferral
- Excessive Collisions
- Late Collision
- FIFO Underrun
- Signal Quality Error

In accordance with the OPB specifications, an early packet termination causes the master to read the packet's status and end the transmission, and the slave is allowed to initiate an early packet termination only after the master has written the control information to the slave or after the master has asserted the DSCR_NOT_VALID signal. These are implemented in both the TAH-MAL EOPB and the TAH-EMAC EOPB.

The EMAC automatically retransmits frames collided on the MII/GMII interface. The Transmit FIFO always preserves the first 64 (in 10/100 Mbps) or 512 (in Gigabit Ethernet media) bytes of the packet until it receives an indication that the collision window has elapsed. Otherwise, if collision was detected within the collision window, the frame is retransmitted automatically without a new request applied to the TAH. The collision information is reported in the status in the form of "Single Collision" and "Multiple Collision". Since these occurrences are not fatal, when TCP segmentation is enabled, these occurrences will not be reported.

30.2.3.8 Empty Packet

The EMAC treats empty packet as if a normal packet had been written, but without writing any data to the FIFO. A valid status word with all zeros will be returned after an empty packet. If hardware acceleration is not enabled, the TAH will act as a passive conduit and will pass anything between the MAL and the EMAC, including empty packets. If hardware acceleration is enabled, empty packets will result in an error, since the fields necessary to carry out the hardware acceleration function are missing.

The EMAC expects that for quadword-aligned packets, the EOPB master activates the related indication about quadword transfer during the last data transfer, rather than providing an empty packet indication. This is indeed the case for the TAH, and the TAH expects the same from the MAL.

30.2.4 TAH Receive Operation

The receive part of the TAH is responsible for transferring packets from the EMAC to the MAL. If hardware acceleration is enabled, the TAH will also verify the TCP/UDP/IP checksum. At the end of the reception process, the TAH will insert the checksum status, if any, to the status/error word provided by the EMAC before presenting the status to the MAL.

30.2.4.1 Normal Operation

The EMAC initiates request for service from the TAH by asserting EMC_RX_FRAME, and switches the EMC_RX_ARB_LEVEL bus to a (non-idle) request level. As a response, the TAH will assert TAH2MAL_RX_FRAME and drive the TAH2MAL_RX_ARB_LEVEL bus to a (non-idle) request level. The MAL fetches the buffer descriptor from memory and presents the Control information to the TAH by asserting the MAL_RX_ACTIVE signal and the MAL_RX_SELECT signal, and deasserting the MAL_RX_OPB_RNW signal to indicate a write transfer. The control information is passed onto the EMAC by the TAH using the same protocol. During MAL transactions, the low order 3 bits of OPB_ABUS is used for transaction qualifiers. The two settings defined provide for the transfer of control/status information and data.

The EMAC signals its readiness to transfer data to the TAH by driving the EMC_RX_ARB_LEVEL bus to a non-idle request level. Likewise, the TAH will signal its readiness to transfer data to the MAL by driving TAH2MAL_RX_ARB_LEVEL to a non-idle request level. Once the TAH has secured the MAL EOPB by observing that the MAL has asserted MAL_RX_ACTIVE, MAL_RX_SELECT, and MAL_RX_OPB_RNW, the TAH will likewise assert TAH2EMAC_RX_ACTIVE, TAH2EMAC_RX_SELECT, and TAH2EMAC_RX_OPB_RNW. Data is transferred from the EMAC to the TAH and onto the MAL. Along the way the TAH will compute the checksum if so specified in the Mode Register.

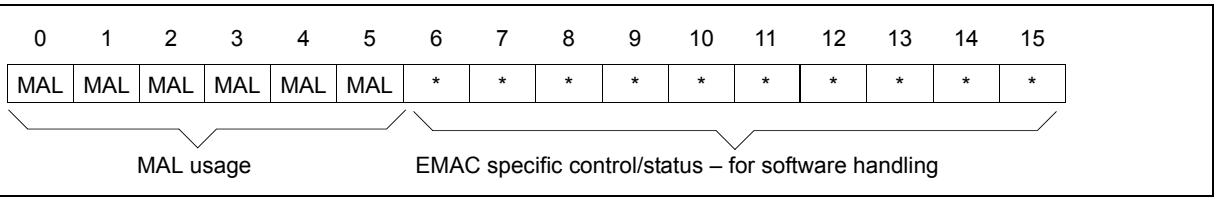
The TAH by itself has no constraint on burst boundaries since it is just a passive conduit of data between the EMAC and the MAL. However, since the MAL has a 16-cycle burst boundary, the TAH must assure that the MAL is available to receive data before accepting data from the EMAC. This is achieved by deasserting the control signals to the EMAC after every 16 cycles and monitoring the same control signals from the MAL. If the control signals are still asserted by the MAL, meaning that the MAL is able to continue the transfer, then the TAH will reassert its control signals to the EMAC. Otherwise transfer from the EMAC is put on hold until the MAL is ready to accept more data.

30.2.4.2 MAL RX Descriptor Status

The EMAC reports status to the TAH by driving EMC_RX_ARB_LEVEL to the highest level. Status is transferred when the EOPB master asserts the ACTIVE and the SELECT signal.

For each packet that is received, the MAL obtains status from the EMAC after reception is complete, and writes this information into the buffer descriptor status/control field. Software uses this information to monitor the status of received packets. See *Buffer Descriptor (BD) Structure* on page 954 for more information on the buffer descriptor structure.

Figure 30-5. MAL RX Descriptor Control/Status Field



Bits	Bit Name	Bit Description	Mode
0:5	Reserved	Always zero	
RX Status Information (Read Access)			
6	Overrun Error	0 No overrun error. 1 The EMAC detected an overrun error. An overrun error occurs if the flow of received data to the RX FIFO is corrupted because of insufficient empty space.	R

Bits	Bit Name	Bit Description	Mode
7	Pause Packet	0 Received packet is not a control pause packet. 1 Received packet is a control pause packet.	R
8	Bad Packet	0 No packet errors. 1 Early termination caused by packet error.	R
9	Runt Packet	0 Duration of PHY_RX_DV signal OK. 1 Duration of PHY_RX_DV signal greater than ShortEventMax Time constant and less than collision windows.	R
10	Short Event	0 Duration of PHY_RX_DV signal OK. 1 Duration of PHY_RX_DV signal was less than ShortEventMaxTime constant	R
11	Alignment Error	0 Received packet length OK. 1 Received packet length not an integral number of octets.	R
12	Bad FCS	0 FCS OK. 1 The FCS value does not match the FCS value calculated by the EMAC.	R
13	Packet Too Long	0 Received packet length OK. 1 Received packet length exceeds maximum packet length. 1518 octets for standard packet. 1522 octets for VLAN tagged packet. 9018 octets for standard jumbo packet. 9022 octets for VLAN tagged jumbo packet. Data following the maximum packet length is not transferred to the MAL	R
14:15	EMAC Errors	00 No errors 01 In range error 10 Out of range error 11 Checksum error if TCP/IP/UDP; otherwise checksum is not verified due to one of the following conditions: 1. Zero checksum for IPv6 2. Packet length mismatch 3. Unsupported header type. **Note:** The TAH reports checksum related errors only if checksum verification is enabled in the mode register (TAHx_MR[CVR] = 1). The TAH will not report checksum related errors if "In range error" or "Out of range error" is reported by the EMAC.	R

30.2.4.3 Normal Packet Ending

For the last data cycle, the EMAC asserts EMC_RX_LAST_DATA and indicates the valid bytes on the data bus using the EMC_RX_BE_EXT_ACK signal. The TAH will repeat these signals on the MAL EOPB using TAH2MAL_RX_LAST_DATA and TAH2MAL_RX_BE_EXT_ACK signals. Following the last data cycle, both the EMAC and the TAH deasserts the Frame signal.

30.2.4.4 Early Packet Termination

The TAH does not terminate packet transmission prematurely. If the EMAC exercises early packet termination, the error status will be reported to the MAL in the same way as for the no error case.

30.2.5 VLAN Support

The TAH is able to handle VLAN tagged frames as specified in IEEE P802.3ac.

VLAN tagged frame is an extension of the standard MAC frame. The extension for VLAN tag support consists of a 4-octet VLAN tag inserted between the end of the Source Address and the beginning of the Length/Type field of the MAC frame. This tag consists of two fields.

1. A 2-octet constant Type field value equal to the VLAN Tag Protocol Identifier (0x8100).
2. A 2-octet field containing Tag Control Information (TCI).

Following the VLAN tag is the MAC Client Data and FCS fields of the basic MAC frame. The length of the frame is extended by four octets by the VLAN tag (up to 1522 bytes maximum, and up to 9022 bytes for jumbo operation). The FCS is calculated over all fields from the Destination Address through the end of the MAC client data or Pad (if present); that is, all fields except the preamble, SFD, and FCS).

30.2.6 Jumbo Frame Support

The TAH supports Ethernet II format Jumbo frames. This format does not change the definition of any Ethernet or TCP/IP fields. The difference between a standard Ethernet II frame and a Jumbo frame is in the value of the IP length field. The use of Jumbo frames is indicated by the choice of the segment size selected. The TAH can support frame sizes up to 16K bytes provided the transmit FIFO is big enough.

30.2.7 Error Handling

During the transmit operation, when an error is detected by the TAH and hardware acceleration is enabled, the transfer will be terminated. If the command/data has been received from the MAL but has not been transferred to the EMAC, then a null transfer to the EMAC will be initiated to solicit any previous status outstanding. The TAH will set bit 7 of the MAL TX Status port to indicate the error. In addition, an interrupt will be generated if interrupt is not disabled, and the error status is saved in the Transmit Status Register until it is read.

During the receive operation, when an error is detected during checksum verification, the error status is reported in the MAL RX Status port but the transfer is not terminated.

30.2.8 Enabling Hardware Acceleration

Hardware accelerate functions can be enabled on a per-packet basis for transmit operations, and on a per EMAC/TAH basis for receive operations. For each transmitted packet, the MAL uses the descriptor control/status field of the buffer descriptor to provide control information to the TAH. Hardware acceleration is enabled by setting the hardware acceleration bits (bits 12–14 as described in *MAL TX Descriptor Control/Status Field* on page 1076) in the buffer descriptor to a non-zero value.

Note: If the hardware accelerate bits are not set to 000 (some form of hardware accelerate is enabled), bits 6 and 7 must also be set. Failure to do so will cause improper operation. If "Generate FCS" is not set, the FCS will be bad, and the EMAC will set bit 6 "Bad FCS on transmitted frame". If "Generate padding" is not set, the frame might not meet the minimum packet size requirement of the network. If this were the case, when the frame is received by the other side, the frame would be considered a “runt frame” and will be reported as such or discarded.

30.3 TAH Registers

This section describes the TAH internal registers. The TAH registers are accessed through the OPB. Any access to the registers should be fullword aligned. Read operations from unused addresses return zeros; write operations to these addresses have no effect. All registers are cleared during power up or when the soft reset is issued except as noted.

There are two TAHs, designated 0 and 1. In the following sections, the registers are specified with a generic name where x represents 0 or 1. TAH registers are accessed at 0x4 EF60_1nYY where n = 3 for TAH0 and n = 4 for TAH1.

Table 30-4. TAH Register Summary

Mnemonic	Register	Address	Access	Page
TAHx_REVID	Revision ID Register	0x4 EF60 1n50	R/W	1083
TAHx_MR	Mode Register	0x4 EF60 1n60	R/W	1083
TAHx_SSR0	Segment Size Register 0	0x4 EF60 1n64	R/W	1085
TAHx_SSR1	Segment Size Register 1	0x4 EF60 1n68	R/W	1085
TAHx_SSR2	Segment Size Register 2	0x4 EF60 1n6C	R/W	1085
TAHx_SSR3	Segment Size Register 3	0x4 EF60 1n70	R/W	1085
TAHx_SSR4	Segment Size Register 4	0x4 EF60 1n74	R/W	1085
TAHx_SSR5	Segment Size Register 5	0x4 EF60 1n78	R/W	1085
TAHx_TSR	Transmit Status Register	0x4 EF60 1n7C	R	1085

30.3.1 TAH Revision ID Register (TAHx_REVID)

TAHx_REVID is the revision ID register. This is a read only register containing the revision number and the branch revision number of the core. The values change with each new revision of the core. *Figure 30-6* describes the TAHx_REVID register bits.

Figure 30-6. TAHx Revision ID Register (TAHx_REVID)

0:11		Reserved	
12:23	RN	Revision Number	Set to 3
24:31	BRN	Branch Revision Number	Set to 0

30.3.2 TAH Mode Register (TAHx_MR)

The TAH mode register defines the configuration modes of the TAH that may be changed at any time during TAH operation. When TAHx_MR[CVR] = 1, TAHx verifies the checksum for IP, TCP and UPD frames. When TAHx_MR[SR] = 1 TAHx activates a soft reset and the bit is cleared after reset is completed.

TAHx_MR[ST] defines the number of bytes (in multiple of 256B) that must be received from the MAL when hardware assist is disabled before the send data is forwarded to the EMAC. For example, a value of 2 means that 512B would have to be fetched from the MAL before the data is sent to the EMAC. A larger value would decrease the chance of underrun in the EMAC but increase the latency. A smaller value would increase the chance of underrun in the EMAC but decrease the latency. A value of 0 means that as soon as data is received from the MAL, it is forwarded to the EMAC. But since transfer to the EMAC does not start until the TAH has accumulated a full burst (16 cycles, or 256B), a value of 0 is functionally equivalent to a value of 1 in TAHx_MR[ST].

TAHx_MR[TFS] defines the size of the transmit FIFO attached to TAHx in units of 2KB. For example, a value of 001 means a 2KB FIFO is present, and 101 means a 10KB FIFO is present. If hardware segmentation is enabled, the size of the transmit FIFO must be bigger than the size of the largest segment by 512B. If hardware checksum is enabled, the size of the transmit FIFO must be bigger than the size of the packet by 80 bytes; otherwise the TAH terminates the transfer with error if hardware acceleration is enabled. For standard ethernet packets, a 2KB transmit FIFO is sufficient. For Alteon style Jumbo packets with a 9KB frame size, a 10KB transmit FIFO size is sufficient.

Figure 30-7. TAH Mode Register (TAHx_MR)

0	CVR	Checksum Verification on Receive 0 Checksum for IP, TCP and UDP packets disabled 1 Checksum for IP, TCP and UDP packets enabled	
1	SR	TAH Software Reset 0 TAH reset is complete 1 Reset the TAH	This bit is self-clearing. It always reads back as zero.
2:7	ST	Send Threshold 000000 - 256 bytes 000001 - 256 bytes 000010 - 512 bytes	
8:10	TFS	Transmit FIFO Size 001 Transmit FIFO size is 2Kbyte 010 Transmit FIFO size is 4Kbyte 011 Transmit FIFO size is 6Kbyte 100 Transmit FIFO size is 8Kbyte 101 Transmit FIFO size is 10Kbyte	The default is 001. Maximum of 10KB FIFO is used.
11	DTFP	Disable Transmit FIFO Parity Protection 0 TAH checks for parity errors in the transmit FIFO 1 TAH does not check for parity errors in the transmit FIFO	
12	DIG	Disable Interrupt Generation 0 TAH generates interrupts when errors are detected 1 TAH does not generate interrupts when errors are detected	
13	IPV6	TAH IPv6 Mode 0 Supports only IPv4 packets (Default) 1 Enable Support for both IPv6 and IPv4 packets	IPv4 mode is always enabled, but IPv6 is controlled by this bit
14	V6INV4	TAH Software Reset 0 No Support for IPv6 within IPv4 (Default) 1 Support IPv6 packets within IPv4	Receive mode only. IPv4 mode is always enabled, but IPv6-within-IPv4 is controlled by this bit
15	DCPPFS	DCR check packet FSM states 0 Current packet FSM transitions (Default) 1 Previous packet FSM transitions	
16	RIP4OF	Retain IPv4 functionality for IPv6-within-IPv4 packets 0 IPv4 functionality (Default) 1 IPv6-within-IPv4 code	Receive mode only. This is for internal Debug
17:31		Reserved	

30.3.3 TAH Segment Size Registers 0:5 (TAHx_SSR0–TAHx_SSR5)

The TAH segment size registers specify the size of the segment to be used when TCP segmentation is enabled. The size (defined as number of halfwords) of a segment refers to the total number of bytes following the type/length field and before the FCS field in an ethernet packet. It includes the IP headers, the TCP headers, and the data payload. For an IEEE 802 formatted packet, it also includes the 8-byte Logical Link Control (LLC) header and Sub-Network Access Protocol (SNAP) header. It does not include the 6-byte MAC destination address, the 6-byte MAC source address, the 2-byte Type/Length field, and the 4-byte FCS field. It also does not include the 4-byte VLAN tag field. For a standard Ethernet packet, the segment size should be set to 1500. For an Alteon style Jumbo frame, the segment size should be set to 9000.

Note: The TAH can accommodate any arbitrary segment size provided that Transmit FIFO is big enough. All segment size registers are initialized to a value of 750, which corresponds to 1500 bytes (size of a standard Ethernet packet).

Figure 30-8. TAH Segment Size Register 0:5 (TAHx_SSR0–TAHx_SSR5)

0:1		Reserved	
2:14	SS	Segment Size Defines the size of the segment in multiples of 2 bytes to be used when TCP segmentation is enabled.	
15:31		Reserved	

30.3.4 TAH Transmit Status Register (TAHx_TSR)

The TAH Transmit Status Register contains the error status that results in the abnormal termination of a transmit operation when hardware acceleration is enabled. The register can only be written by the TAH. The content is cleared when the Transmit Status Register is read.

When hardware checksum is enabled, the size of the transmit FIFO is not more than 80 bytes bigger than the size of the packet. When hardware segmentation is enabled, the size of the transmit FIFO is no more than 512 bytes bigger than the size of a segment.

Figure 30-9 describes the TAHx_TSR register bits.

Figure 30-9. TAH Transmit Status Register (TAHx_TSR)

0	TFTS	Transmit FIFO Too Small The size of the transmit FIFO is not more than 80bytes bigger than the size of the packet when hardware checksum is enabled; or the size of the transmit FIFO is no more than 512 bytes bigger than the size of a segment when hardware segmentation is enabled.	
1	UH	Unrecognized Header The packet is in a format that is not supported by hardware	
2	NIPF	Not IP Version 4 Format The packet is not in IP version 4 format	
3	IPOP	IP Option Present IP option is present in the packet	

4	NISF	No IEEE SNAP Format The packet uses the IEEE 802 format but the SNAP header is incorrect	
5	ILTS	IP Length Too Short The total length in the IP header is smaller than 40 bytes.	
6	IPFP	IP Fragment Present The packet is part of an IP fragment.	
7	UP	Unsupported Protocol The protocol in use is neither TCP or UDP or IPv6withinIPv4 (Protocol x29 IPv6withinIPv4 is controlled by mode register bit)	
8	TFP	TCP Flags Present Some TCP flags other than ACK are set.	
9	SUDP	Segmentation for UDP UDP packets cannot be segmented.	
10	DLM	Data Length Mismatch Amount of send data is smaller than the size of the TCP/IP header or the value in the IP length field.	If more send data is present, the extra data will be discarded with no error.
11	SIEEE	Segmentation for IEEE The size of the segment exceeds 1500 bytes for IEEE formatted packets when hardware segmentation is enabled.	
12	TFPE	Transmit FIFO Parity Error Parity error is detected in the Transmit FIFO.	
13	SSTS	Segment Size Too Small The size of the segment is less than 168 bytes for DIX formatted segments or 176 bytes for IEEE formatted segments when hardware segmentation is enabled.	
14	NIP6F	Not IP Version 6 Format The packet is not in IP version 6 format.	
15	IP6OP	IPv6 Option Present IPv6 option is present in the packet.	
16	IP6EHP	IPv6 Extended Header Present IPv6 extended header is present in the packet.	
17	IP6UNH	IPv6 Unsupported Protocol The protocol in use is neither TCP or UDP.	
18	IP6HPLM	IPv6 Header Payload Length Mismatch Payload length in IPv6 Header mismatches the UDP Header Payload Length.	
19:31		Reserved	

30.4 Power-Up and Initialization

To reset or reconfigure the TAH, both the EMAC and the MAL must be in the quiescent state to avoid erroneous situations. There are two types of reset operations that may be applied to the EMAC: hard reset and soft rest.

30.4.1 Hard Reset

When RESET input is asserted, the TAH aborts all on-going activities unconditionally, initializes all internal state machines, counters, and registers. The reset signal must be asserted for at least two cycles of the slowest clock domain inside the TAH in order to be properly recognized. This means the hard reset must be at least 800 ns.

30.4.2 Soft Reset

The software should first reset the appropriate channels in the MAL (see *MAL Configuration Register (MAL0_CFG)* on page 977) and then initialize a soft reset by setting the Soft Reset bit in Mode Register to 1. As a response to the soft reset, the TAH aborts all on-going activities unconditionally, initializes all internal state machines, counters, and registers. After the TAH finishes all activities related to the soft reset processing, it clears the Soft Reset bit.

31. UART Serial Port Operations

The PPC460EX/EXr/GT contains four universal asynchronous receiver/transmitters (UARTs) to support communications with serial peripheral devices. Each UART includes a 64-byte send and a 64-byte receive FIFO.

Features of the UART include:

- 64-byte send FIFO, 64-byte receive FIFO
- Full duplex operation
- Programmable baud rate generator
- Supports 5- to 8-bit word size, 1 or 2 stop bits, even, odd, or no parity
- One 8-wire interface, two 4-wire interfaces, one 4-wire interface and two 2-wire interfaces, or four 2-wire interfaces
- Hardware flow control is selectable

The UART performs serial-to-parallel conversion on data characters received from a peripheral device, and parallel-to-serial conversion on data characters received from the processor. The processor can read the complete status of the UART at any time during the functional operation. Status information reported includes the type and condition of the transfer operations being performed by the UART, as well as any error conditions, such as parity, overrun, framing, and break interrupt.

This UART powers up in character mode and can be put into FIFO mode to relieve the processor of excessive software overhead. Here, internal FIFOs are activated allowing 64 bytes (plus 3 bits per byte of error data in the RCVR FIFO) to be stored in both receive and transmit modes.

The source of the UART serial clock input is selected in SDR0_UART0:3[U0EC:U3EC] bits 8. Either the internal serial clock or an external serial clock can be selected. A programmable baud rate generator is included that is capable of dividing the UART serial clock input by a divisor of 1 to (2^{16} – 1) and producing the 16× clock required for driving the UART internal transmitter/receiver logic. The internal serial clock input is derived from the PLB clock by a divisor specified in SDR0_UART0, 1, 2 and 3[UDIV]. See *Clocking* on page 237 for additional information.

The UART has an interrupt system that can be programmed to the user's requirements, helping to minimize the computing required to handle the communications link. UART interrupts are capable of triggering an interrupt request to the PPC460EX/EXr/GT interrupt controller.

31.1 Functional Description

- Equipped with complete status reporting capability
- Transmitter and receiver are each buffered with 64-byte FIFOs when FIFO mode selected
- Can add/delete standard asynchronous communication bits such as start, stop, and parity to/from the serial data
- When in character mode, holding and shift registers eliminate the need for precise synchronization between the processor and serial data
- Full prioritized interrupt system controls
- Independently controlled transmit, receive, line status, and data set interrupts
- Programmable baud rate generator divides the UART serial clock input by 1 to (2^{16}–1) and generates the 16x clock:

  Baud rate (bps) = (Serial Clock Input) / (16 x Decimal Divisor)

- Receiver uses 5-way oversampling as follows: it samples each serial bit five times, and if at least three of the samples are 1's, the bit is determined to be a 1, otherwise it is a 0
- Fully programmable serial-interface characteristics:
  - 5-, 6-, 7-, or 8-bit characters
  - Even, odd, or no parity bit generation and detection
  - 1-,1.5-, or 2-stop bit generation
  - Variable baud rate
- Line break generation and detection, and false start bit detection
- Internal diagnostic capability:
  - Loopback controls for communications link fault isolation
  - Break, parity, overrun, framing error simulation

31.2 Serial Port Clocking

Each of the four PPC460EX/EXr/GT UARTs can be clocked individually from an external serial clock or from a single internally generated serial clock. The internally generated serial clock is derived from the PLB clock, and can only be an integer (n) fraction of that clock where n is in the range from 1 to 256.

The formula for the serial clock and bit rate when using the PLB as the clock source is:

PLB:UART Divisor = SDR0_UARTx[UxDIV]

Serial Clock = PLBClk/PLB:UART Divisor

UART Divisor = UARTx_DLM | UARTx_DLL

Bit Rate = Serial Clock/(16* UART Divisor)

The choice of serial clock frequency affects the serial communications error rate. If an external clock of 1.8432 MHz (or some multiple of this frequency) is used, the error rate approaches zero. However the internally generated clock operates only at certain clock frequencies, all of which result in a small, non-zero error rate.

The formula for the serial clock and bit rate when using the external serial clock, UARTSerClk, as the clock source is:

Serial Clock = UARTSerClk

UART Divisor = UARTx_DLM || UARTx_DLL

Bit Rate = Serial Clock/(16* UART Divisor)

Acceptable baud rates are always integral multiples of 300 (for example, 1200 = 4 × 300). *Table 31-1* shows optimum UART divisor and divide ratios for a range of possible baud rates. This information is provided for various clock frequencies. Note that the serial clock frequency must be less than half the OPB frequency. The UART divisor is programmed in UARTx_DLM and UARTx_DLL (see *Divisor Latch LSB and MSB Registers (UARTx_DLL and UARTx_DLM)* on page 1101). The value range is 1 to (2^{16}–1) = 65535.

Table 31-1. Bit Rate Settings (Sheet 1 of 3)

Desired Bit Rate (bps)	PLB:UART Divide Ratio	Serial Clock Frequency (MHz)	Minimum OPB Frequency (MHZ)	UART Divisor	Actual Bit Rate (bps)	Error (%)
PLB Clock =133.33 MHz						
1200	14	9.5238	19.4175	496	1200.07	0.006
2400	14	9.5238	19.4175	248	2400.15	0.01
4800	14	9.5238	19.4175	124	4800.31	0.01
9600	14	9.5238	19.4175	62	9600.61	0.01
19200	14	9.5238	19.4175	31	19201.23	0.01
28800	17	7.8431	15.0376	17	28835.06	0.12
33600	31	4.3011	8.4033	8	33602.15	0.01
38400	31	4.3011	8.4033	7	38402.46	0.01
57600	29	4.5977	9.2807	5	57471.26	-0.22
115200	24	5.5556	11.2359	3	115740.74	0.47
307200	27	4.9383	9.9751	1	308641.97	0.47
PLB Clock =155.56 MHz						
1200	37	4.2042	8.4084	219	1199.83	-0.01
2400	30	5.1852	10.3704	135	2400.55	0.02
4800	25	6.2222	12.4444	81	4801.10	0.02
9600	22	7.0707	14.1414	46	9606.94	0.07
19200	22	7.0707	14.1414	23	19213.88	0.07
28800	26	5.9829	11.9658	13	28763.97	-0.13
33600	17	9.1503	18.30007	17	33640.91	0.12
38400	31	8.6022	17.2043	14	38402.46	0.01
57600	13	11.9658	23.9316	13	57527.94	-0.13
115200	21	7.4074	14.8148	4	115740.74	0.47
307200	16	9.7222	19.4444	2	303819.44	-1.10

Table 31-1. Bit Rate Settings (Sheet 2 of 3)

Desired Bit Rate (bps)	PLB:UART Divide Ratio	Serial Clock Frequency (MHz)	Minimum OPB Frequency (MHZ)	UART Divisor	Actual Bit Rate (bps)	Error (%)
PLB Clock = 160.00 MHz						
1200	18	8.8889	17.7778	463	1199.90	-0.01
2400	17	9.4118	18.8235	245	2400.96	0.04
4800	20	8.0000	16.0000	104	4807.69	0.16
9600	20	8.0000	16.00	52	9615.38	0.16
19200	20	8.0000	16.00	26	19230.77	0.16
28800	29	5.5172	11.0345	12	28735.63	-0.22
33600	27	5.9259	11.8519	11	33670.03	0.21
38400	20	8.0000	16.0000	13	38461.54	0.16
57600	29	5.5172	11.0345	6	57471.26	-0.22
115200	29	5.5172	11.0345	3	114942.53	-0.22
307200	33	4.8485	9.6970	1	303030.30	-1.36
PLB Clock = 166.66 MHz						
1200	31	5.3763	10.8695	280	1200.08	0.01
2400	31	5.3763	10.8695	140	2400.15	0.01
4800	31	5.3763	10.8695	70	4800.31	0.01
9600	31	5.3763	10.8695	35	9600.61	0.01
19200	32	5.2083	10.5263	17	19148.28	-0.27
28800	19	8.7719	17.8571	19	28855.03	0.19
33600	31	5.3763	10.8695	10	33602.15	0.01
38400	17	9.8039	20.0000	16	38296.57	-0.27
57600	30	5.5556	11.2359	6	57870.37	0.47
115200	30	5.5556	11.2359	3	115740.74	0.47
307200	17	9.8039	20.0000	2	306372.55	-0.27

Table 31-1. Bit Rate Settings (Sheet 3 of 3)

Desired Bit Rate (bps)	PLB:UART Divide Ratio	Serial Clock Frequency (MHz)	Minimum OPB Frequency (MHZ)	UART Divisor	Actual Bit Rate (bps)	Error (%)
PLB Clock = 177.78 MHz						
1200	20	8.8889	17.7778	463	1199.90	-0.01
2400	26	6.8376	13.6752	178	2400.85	0.04
4800	26	6.8376	13.6752	89	4801.69	0.04
9600	34	5.2288	10.4575	34	9611.69	0.12
19200	34	5.2288	10.4575	17	19223.38	0.12
28800	35	5.0794	10.1587	11	28860.03	0.21
33600	30	5.9259	11.8519	11	33670.03	0.21
38400	29	6.1303	12.2605	10	38314.18	-0.22
57600	24	7.4074	14.8148	8	57870.37	0.47
115200	24	7.4074	14.8148	4	115740.74	0.47
307200	18	9.8765	19.7531	2	308641.98	0.47
PLB Clock = 200.00 MHz						
1200	21	9.5238	19.04776	496	1200.08	0.01
2400	21	9.5238	19.04776	248	2400.15	0.01
4800	21	9.5238	19.04776	124	4800.31	0.01
9600	21	9.5238	19.04776	62	9600.61	0.01
19200	21	9.5238	19.04776	31	19201.23	0.01
28800	31	6.4516	12.9032	14	28801.84	0.01
33600	31	6.4516	12.9032	12	33602.15	0.01
38400	25	8.0000	16.0000	13	38461.54	0.16
57600	31	6.4516	12.9000	7	57603.69	0.01
115200	27	7.4074	14.8148	4	115740.74	0.47
307200	41	4.8780	9.7561	1	304878.05	-0.76

31.3 UART Registers

In the PPC460EX/EXr/GT there are four UARTs, designated 0 through 3. In the following sections, the registers are specified with a generic name where x represents 0, 1, 2, or 3. For example, the Line Control Register appears as a UARTx_LCR.

UART registers are accessed at 0x4 EF60_0nYY where n = 3 for UART0, n = 4 for UART1, n = 5 for UART2, and n = 6 for UART3.

Table 31-2. UART Configuration Registers

Mnemonic	Register	Address	Access	Page
UARTx_RBR	UART x Receiver Buffer Register	0x4 EF60 0n00 [1]	R	1095
UARTx_THR	UART x Transmitter Holding Register	0x4 EF60 0n00 [1]	W	1095
UARTx_IER	UART x Interrupt Enable Register	0x4 EF60 0n01 [1]	R/W	1095
UARTx_IIR	UART x Interrupt Identification Register	0x4 EF60 0n02	R	1096
UARTx_FCR	UART x FIFO Control Register	0x4 EF60 0n02	W	1097
UARTx_LCR	UART x Line Control Register	0x4 EF60 0n03	R/W	1097
UARTx_MCR	UART x Modem Control Register	0x4 EF60 0n04	R/W	1098
UARTx_LSR	UART x Line Status Register	0x4 EF60 0n05	R/W	1099
UARTx_MSR	UART x Modem Status Register	0x4 EF60 0n06	R/W	1100
UARTx_SCR	UART x Scratch Register	0x4 EF60 0n07	R/W	1101
UARTx_DLL	UART x Divisor Latch (LSB)	0x4 EF60 0n00 [1]	R/W	1101
UARTx_DLM	UART x Divisor Latch (MSB)	0x4 EF60 0n01 [1]	R/W	1101

Note 1: UARTx_LCR[DLAB] controls the function accessed through registers 0x4 EF60_0n00 and 0x4 EF60_0n01. When UARTx_LCR[DLAB] is 0, access is enabled to the Receiver/Transmitter registers and the Interrupt Enable register. When UARTx_LCR[DLAB] is 1, access is enabled to the Divisor Latch registers.

In the four wire interfaces, two of the four wires are Tx and Rx. The remaining two wires can be programmed as a combination of DTR and DSR, or CTS and RTS. The selection is made in SDR0_PFC1[U0ME] and SDR0_PFC1[U1ME] respectively. DCD and RI are not available on the 4-wire interface.

31.3.1 Receiver Buffer Registers (UARTx_RBR)

Figure 31-1. UART Receiver Buffer Registers (UARTx_RBR)

0:7		Data bit	

Note: UARTx_RBR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.

31.3.2 Transmitter Holding Registers (UARTx_THR)

Figure 31-2. UART Transmitter Holding Registers (UARTx_THR)

0:7		Data bit	

Note: UARTx_THR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.

31.3.3 Interrupt Enable Registers (UARTx_IER)

Five UART interrupts on four priority levels are enabled via the Interrupt Enable Register, UARTx_IER. Any of the five interrupts can be used to surface a UART interrupt to the PPC460EX/EXr/GT interrupt controller. Each interrupt can be enabled by setting its appropriate bit. Resetting UARTx_IER[4:7] totally disables the UART interrupt system. Disabling an interrupt prevents it from being shown as active in the UARTx_IIR and prevents it from signaling a UART interrupt to the PPC460EX/EXr/GT interrupt controller. See *Table 31-3 Interrupt Priority Level* on page 1096.

Figure 31-3. UART Interrupt Enable Registers (UARTx_IER)

0:3		Reserved	Always 0.
4	EDSSI	Modem Status Interrupt 0 Disable modem status interrupt 1 Enable modem status interrupt	
5	ELSI	Receiver Line Status Interrupt enable 0 Disable receiver line status interrupt 1 Enable receiver line status interrupt	
6	ETBEI	Transmitter Holding Register Empty Interrupt enable 0 Disable transmitter holding register empty interrupt 1 Enable transmitter holding register empty interrupt	
7	ERBFI	Received Data Available Interrupt enable 0 Disable received data available interrupt 1 Enable received data available interrupt	In FIFO mode, timeout interrupts follow the enable/disable state of ERBFI.

Note: UART0_IER is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.

31.3.4 Interrupt Identification Registers (UARTx_IIR)

The UART prioritizes interrupts into four levels which are recorded in the Interrupt Identification Register. The interrupt types in the order of their priority are as follows:

1. Receiver line status
2. Received data available and character timeout indication
3. Transmitter holding register empty
4. Modem status

Table 31-3. Interrupt Priority Level

IIR Bit 4	IIR Bit 5	IIR Bit 6	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	1	1	1	Receiver Line Status	Overrun, Parity or Framing Error, or Break Interrupt.	Read LSR.
0	1	0	2	Received Data Available	Receiver data available or trigger level reached.	Read RBR, or FIFO drops below trigger level.
1	1	0	2	Character Timeout Indication	No characters have been removed from or input to the receiver FIFO during the last four character times and it contains at least one character during this time.	Read RBR.
0	0	1	3	Transmitter Holding Register Empty	Transmitter Holding Register Empty.	Read IIR (if source of interrupt) or write THR.
0	0	0	4	Modem Status	Clear to Send, Data Set Ready, Ring Indicator or Data Carrier Detect.	Read MSR.

When the processor accesses UARTx_IIR, the UART records new interrupts, but does not change its current contents until the access by the processor is complete. The UART indicates the highest priority interrupt pending to the PPC460EX/EXr/GT interrupt controller via the IIR.

Figure 31-4. UART Interrupt Identification Registers (UARTx_IIR)

0:1	FE	FIFO Control Indicator 00 FIFOs disabled (UARTx_FCR[FC] = 0) 01 Reserved 10 Reserved 11 FIFOs enabled (UARTx_FCR[FC] = 1)	
2:3		Reserved	
4:6	INTID	Interrupt Priority Level 000 Priority level 4 001 Priority level 3 010 Priority level 2 011 Priority level 1 100 Reserved 101 Reserved 110 Priority level 2 111 Reserved	**Note:** Priority 1 is highest priority.

7	INTP	Interrupt Pending 0 Interrupt is pending 1 No interrupt pending	When set to 0, IIR contents can be used as a pointer to the appropriate interrupt service routine.
Note: UARTx_IIR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

31.3.5 FIFO Control Registers (UARTx_FCR)

The FIFO control register has the same address as the IIR and is a write-only register. This register is used to perform FIFO control operations such as selecting the type of DMA signaling, setting the receiver FIFO trigger levels, clearing the FIFOs, and enabling the FIFO.

Figure 31-5. UART FIFO Control Registers (UARTx_FCR)

0:1	RFTL	Receiver FIFO Trigger Level 00 01 byte 01 16 bytes 10 32 bytes 11 56 bytes	
2:3		Reserved	
4	DMS	DMA Mode Select 0 Mode 0 = single transfer 1 Mode 1 = multiple transfers	Select single or multiple transfer mode if UARTx_FCR[7] = 1.
5	TFR	Transmitter FIFO Reset 0 Operation complete 1 Reset the transmitter FIFO	A 1 written to this bit clears all bytes in the transmitter FIFO and resets all of its counter logic to 0. The transmitter shift register is not cleared. This bit is self-clearing.
6	RFR	Receiver FIFO Reset 0 Operation complete 1 Reset the receiver FIFO	A 1 written to this bit clears all bytes in the receiver FIFO and resets all of its counter logic to 0. The receiver shift register is not cleared. This bit is self-clearing.
7	FE	FIFO Enable 0 Disable FIFOs 1 Enable FIFOs	When set to 1, both the receiver and transmitter FIFOs are enabled. When set to 0, both receiver and transmitter FIFOs are reset. Data is automatically cleared from both FIFOs when changing to and from FIFO and 16450 modes. Programming other bits will be ignored if this bit is not a 1.
Note: UARTx_FCR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

31.3.6 Line Control Registers (UARTx_LCR)

The system programmer uses the line control register (LCR) to specify the format of the asynchronous data communications exchange and to set the Divisor Latch Access bit. The contents of the LCR can also be read by the processor. The read capability simplifies system programming, and eliminates the need for separate storage of the line characteristics in system memory.

Figure 31-6. UART Line Control Registers (UARTx_LCR)			
0	DLAB	Divisor Latch Access Bit 0 Address RBR, THR and IER with LTADR2-0 for read or write operation 1 Address Divisor Latches with LTADR2-0 for read or write operation	
1	SB	Set Break 0 Disable Break 1 Enable Break	Causes a break condition to be transmitted to the UART when the function is receiving. SOUT is forced to the spacing state (0). This bit acts only on SOUT and has no effect on the transmitter logic.
2	SP	Sticky Parity 0 Disable sticky parity 1 Enable sticky parity	If UARTx_LCR[EPS] = 1 and UARTx_LCR[PEN] = 1, the parity bit is transmitted and checked as 0. If UARTx_LCR [EPS] = 0 and UARTx_LCR[PEN] = 1,the parity bit is transmitted and checked as 1.
3	EPS	Even Parity Select 0 Generate odd parity 1 Generate even parity	This bit is significant only if UARTx_LCR[PEN] = 1.
4	PEN	Parity Enable 0 Disable parity checking 1 Enable parity checking	
5	SBS	Stop Bit Select 0 Characters have 1 stop bit 1 Characters have 1.5 or 2 stop bits	If UARTx_LCR[WPS1,WPS0] = 00, characters have 1.5 stop bits. For any other value of UARTx_LCR[WPS1,WPS0], characters have 2 stop bits. The receiver checks the first stop bit only, regardless of how many stop bits are selected.
6	WLS1	Word Length Select Bits 1 00 Use 5-bit characters 01 Use 6-bit characters 10 Use 7-bit characters 11 Use 8-bit characters	
7	WLS0	Word Length Select Bits 0 00 Use 5-bit characters 01 Use 6-bit characters 10 Use 7-bit characters 11 Use 8-bit characters	
Note: UART0_LCR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

31.3.7 Modem Control Registers (UARTx_MCR)

The interface between the modem, data set, or peripheral device emulating a modem, and the UART, is controlled by the Modem Control Register (UARTx_MCR).

Figure 31-7. UART Modem Control Registers (UARTx_MCR)			
0:1		Reserved	Always 0.
2	AFC	Auto Flow Control 0 Hardware flow control disabled 1 Hardware flow control enabled	

3	LOOP	Loopback Mode 0 Disabled 1 Enabled	Provides a local loop back feature for diagnostic testing of the UART. The following occurs: 1. SOUT is set to the marking state (logic 1) SIN is disconnected. 2. The output of the transmitter shift register feeds the input of the receiver shift register. 3. The four modem control inputs $\overline{DSR}$, $\overline{CTS}$, $\overline{RI}$, and $\overline{DCD}$ are disconnected. 4. The four modem control outputs $\overline{DTR}$, $\overline{RTS}$, $\overline{OUT1}$, and $\overline{OUT2}$ are set to a logic 1 (their inactive state). 5. The four modem control outputs are connected internally to the four modem control inputs. Transmitted data is immediately received to verify the UART transmit and receive data paths. Receiver and transmitter interrupts are operational. Their sources are external to the UART. Also operational are the modem control interrupts, but their source is the low-order 4 bits of UARTx_MCR instead of the modem control inputs to the UART. UARTx_IER still controls the interrupts.
4	OUT2	User Output 2 0 $\overline{OUT2}$ inactive (1) 1 $\overline{OUT2}$ active (0)	The $\overline{OUT2}$ bit may be written and read but it provides no function
5	OUT1	User Output 1 0 $\overline{OUT1}$ inactive (1) 1 $\overline{OUT1}$ active (0)	The $\overline{OUT1}$ bit may be written and read but it provides no function
6	RTS	Request To Send 0 $\overline{RTS}$ inactive (1) 1 $\overline{RTS}$ active (0)	
7	DTR	Data Terminal Ready 0 $\overline{DTR}$ inactive (1) 1 $\overline{DTR}$ active (0)	
Note: UARTx_MCR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

31.3.8 Line Status Registers (UARTx_LSR)

Information concerning the data transfer is held for the processor in this register. Bits 3 through 6 are conditions that produce a receiver line status interrupt whenever the condition corresponding to the active bit is detected and the interrupt is enabled. This register is intended for read operations only and writing is not recommended.

Figure 31-8. UART Line Status Registers (UARTx_LSR)

0	RFE	Receiver FIFO Error Indicator 0 In FIFO mode, reset to 0 when the processor reads the UARTx_LSR, provided there are no subsequent errors in the FIFO. 1 There are one or more instances of parity error, framing error or break indication in the FIFO.	Always 0 in 16450 mode.
1	TEMT	Transmitter Empty Indicator 0 Reset to 0 whenever the THR or the transmitter shift register contain a character. In FIFO mode, it is reset to 0 whenever the transmitter FIFO or the transmitter shift register contain a character. 1 Set to 1 when the THR and the Transmitter shift register are both empty. In FIFO mode, it is set to 1 when the transmitter FIFO and the transmitter shift register are both empty.	

2	THRE	Transmitter Holding Register Empty Indicator 0 Concurrent reset to 0 with the loading of the THR by the processor. In FIFO mode it is reset to 0 when at least one byte is written to the transmitter FIFO. 1 Set to 1 when the UART is ready to accept a new character for transmission. In FIFO mode, this bit is set when the transmitter FIFO is empty.	When UARTx_IER[THRE] = 1, the UART issues an interrupt to the PPC460EX/EXr/GT interrupt controller. This bit is set to 1 when a character is transferred from the THR to the transmitter shift register.
3	BI	Break Interrupt Indicator. 0 Reset to 0 whenever processor reads Line Status Register (LSR). 1 Set to 1 whenever the received data input is held at the spacing level (0) for longer than a full word transmission time.	The full word transmission time is the time required for the start bit, data bits (can be 5–8 bits), parity and stop bits. In FIFO mode, this error is revealed to the processor when the character this error is associated with is at the top of the FIFO. Only one 0 character is loaded into the receiver FIFO when a break occurs. After the next valid start bit is received and has gone into the marking state, the next character transfer is enabled. Error causes a Receiver Line Status Interrupt.
4	FE	Framing Error Indicator. 0 Reset to 0 whenever processor reads LSR. 1 Set to 1 whenever stop bit following the last data bit or parity bit is detected as 0 (spacing level). Indicates that a valid stop bit was not found in the received character.	Error causes a Receiver Line Status Interrupt.
5	PE	Parity Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Indicates that the received data character does not have the correct parity as determined by the even parity select bit (UARTx_LCR.[EPS]). Set to 1 upon detection of a parity error.	In FIFO mode, this error is revealed to the processor when the character this error is associated with is at the top of the FIFO. Error causes a Receiver Line Status Interrupt.
6	OE	Overrun Error Indicator. 0 Reset to 0 whenever processor reads UARTx_LSR. 1 Data in the RBR was read by the processor before the next character was transferred into the UARTx_RBR, hence the original data was lost.	In FIFO mode, if the incoming data continues to fill the FIFO beyond the trigger level, an OE occurs only after the FIFO is completely full and the entire next character has been received in the receiver shift register. The processor is informed of the OE immediately upon occurrence. The character in the shift register will be overwritten and will not be transferred to the FIFO. Error causes a Receiver Line Status Interrupt.
7	DR	Receiver Data Ready Indicator. 0 Reset to 0 when all data has been read from the receiver FIFO or the UARTx_RBR. 1 An entire incoming character has been received into the UARTx_RBR or receiver FIFO.	
Note: UARTx_LSR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

31.3.9 Modem Status Registers (UARTx_MSR)

The processor can monitor the present state of the modem (or peripheral device) control lines by reading the Modem Status Register (UARTx_MSR). In addition, the UARTx_MSR has four bits to indicate if any of the modem (or peripheral device) control lines have changed state.

Figure 31-9. UART Modem Status Registers (UARTx_MSR)

0	DCD	Data Carrier Detect	In loop back mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT2].
1	RI	Ring Indicator	In loop back mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[OUT1].

2	DSR	Data Set Ready	In loop back mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[DTR].
3	CTS	Clear To Send	In loop back mode (UARTx_MCR[LB] is 1), it is equivalent to UARTx_MCR[RTS].
4	DDCD	Delta Data Carrier Detect 0 Set when processor reads the Modem Status Register 1 $\overline{DCD}$ input changed state	Indicates that the $\overline{DCD}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
5	TERI	Trailing Edge of Ring Indicator 0 Set when processor reads the Modem Status Register 1 $\overline{RI}$ input changed from 0 to 1	Indicates that the $\overline{RI}$ input to the UART changed from 0 to 1 since the processor last read the Modem Status Register. A modem status interrupt is generated.
6	DDSR	Delta Data Set Ready 0 Set when processor reads the Modem Status Register 1 $\overline{DSR}$ input changed state	Indicates that the $\overline{DSR}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
7	DCTS	Delta Clear To Send 0 Set when processor reads the Modem Status Register 1 $\overline{CTS}$ input changed state	Indicates that the $\overline{CTS}$ input to the UART has changed state since the processor last read the Modem Status Register. A modem status interrupt is generated.
Note: UARTx_MSR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

31.3.10 Scratchpad Registers (UARTx_SCR)

A scratch pad register intended for use by the programmer as a temporary data location is provided in this UART. It does not control the UART operation in any way.

Figure 31-10. UART Scratch Pad Registers (UARTx_SCR)

0:7		Data bits	
Note: UARTx_SCR is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

31.3.11 Divisor Latch LSB and MSB Registers (UARTx_DLL and UARTx_DLM)

The divisor latches are used to program the UART divisor used in generating the baud clock. A 16-bit divisor may be programmed through these registers. Access to these registers is provided by setting UARTx_LCR[DLAB] = 1. These registers have a power-on reset value of 0.

Figure 31-11. UART Baud-Rate Divisor Latch (MSB) Registers (UARTx_DLM)

0:7		Data bits	
Note: UARTx_DLM is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

Figure 31-12. UART Baud-Rate Divisor Latch (LSB) Registers (UARTx_DLL)

8:15		Data bits	
Note: UARTx_DLL is shown in standard PowerPC bit notation, where 0 is the msb and 7 is the lsb.			

The UART divisor is calculated using the following formula:

UART Divisor = Serial Input Clock/(16 x Bit Rate)

For example, if the serial input clock= 11.0592MHz and a bit rate of 9600 bps is required:

UART Divisor = Serial Input Clock/(16 x Bit Rate)

= 11,059,200/(16x9600)

= 72 = 0x48

For this example, UARTx_DLM should be programmed to 0 and UARTx_DLL register should be programmed to 0x48. Due to the error introduced by rounding, some bit rates cannot be generated at certain serial input clock frequencies. *Table 31-4* lists some common baud rates and their corresponding Divisor Latch register values with a serial input clock of 11.0592MHz.

Table 31-4. Divisor Latch Settings for Certain Baud Rates

Bit Rate (bps)	Divisor Latch MSB	Divisor Latch LSB
9600	0x00	0x48
19200	0x00	0x24
28800	0x00	0x18
38400	0x00	0x12
57600	0x00	0x0C

31.4 UART System Device Control Registers

SDR0_UARTn registers are accessed indirectly through the SDR0_CFGADDR and SDR0_CFGDATA registers using the **mtdcr** and **mfdcr** instructions. *Table 3-1* on page 154 lists the DCR addresses for the SDR0_CFGADDR and SDR0_CFGDATA registers.

To read or write one of the SDR0_UARTn registers, software first writes the register address offset of the target register into the SDR0_CFGADDR register. The target register can then be read or written through the SDR0_CFGDATA DCR register. The following PowerPC code illustrates this procedure by reading the SDR0_UART0 register:

```
li r3, SDR0_UART0_Offset
mtdcr SDR0_CFGADDR, r3      ! write SDR0_CFGADDR  with the SDR0_UART0 DCR offset address
mfdcr r4, SDR0_CFGDATA      ! read the content of SDR0_UART0 from SDR0_CFGDATA
```

Table 31-5. UART SDR Registers

Register	Description	DCR Offset	Access	Page
SDR0_UART0	UART0 Configuration Register	0x0120	R/W	1103
SDR0_UART1	UART1 Configuration Register	0x0121	R/W	1103
SDR0_UART2	UART2 Configuration Register	0x0122	R/W	1104
SDR0_UART3	UART3 Configuration Register	0x0123	R/W	1105

31.4.1 UART Configuration Register 0 (SDR0_UART0)

Reset value = 0x20000026

Figure 31-13. UART Configuration Register 0 (SDR0_UART0)

0:3	U0ICS	UART 0 Internal Clock Source 0000 Reserved 0001 Reserved 0010 UART 0 internal clock source = PLB 0011 Reserved	
4:7		Reserved	
8	U0EC	UART 0 EXT Clock Enable 0 UART0 uses the internal serial clock 1 UART0 uses the external serial clock	
9	U0DTE	UART 0 DMA Transmit Enable 0 UART0 DMA transmit is disabled 1 UART0 DMA transmit is enabled	
10	U0DRE	UART 0 DMA Receive Enable 0 UART0 DMA receive is disabled 1 UART0 DMA receive is enabled	
11	U0DC	UART 0 DMA Clear 0 U0DTE and U0DRE are not cleared when UART receives a corresponding terminal count. 1 U0DTE and U0DRE are cleared when UART receives a corresponding terminal count.	
12:23		Reserved	
24:31	U0DIV	UART 0 Divider 0000_0000 - divider = 256 0000_0001 - divider = 1 0000_0010 - divider = 2 ..... 1111_1111 - divider = 255	The source for SDR0_UART0[U0DIV] is indicated by SDR0_UART0[U0ICS]

31.4.2 UART Configuration Register 1 (SDR0_UART1)

Reset value = 0x20000026

Figure 31-14. UART Configuration Register 1 (SDR0_UART1)

0:3	U1ICS	UART 1 Internal Clock Source 0000 Reserved 0001 Reserved 0010 UART1 internal clock source = PLB 0011 Reserved	
4:7		Reserved	
8	U1EC	UART 1 EXT Clock Enable 0 UART1 uses the internal serial clock 1 UART1 uses the external serial clock	
9	U1DTE	UART 1 DMA Transmit Enable 0 UART1 DMA transmit is disabled 1 UART1 DMA transmit is enabled	

10	U1DRE	UART 1 DMA Receive Enable 0 UART1 DMA receive is disabled 1 UART1 DMA receive is enabled	
11	U1DC	UART 1 DMA Clear 0 U1DTE and U1DRE are not cleared when UART receives a corresponding terminal count. 1 U1DTE and U1DRE are cleared when UART receives a corresponding terminal count.	
12:23		Reserved	
24:31	U1DIV	UART 1 Divider 0000_0000 - UART1 divisor = 256 0000_0001 - UART1 divisor = 1 0000_0010 - UART1 divisor = 2 ..... 1111_1111 - UART1 divisor = 255	The source for SDR0_UART1[U1DIV] is indicated by SDR0_UART1[U1ICS]

31.4.3 UART Configuration Register 2 (SDR0_UART2)

Reset value = 0x20000026

Figure 31-15. UART Configuration Register 2 (SDR0_UART2)			
0:3	U2ICS	UART 2 Internal Clock Source 0000 Reserved 0001 Reserved 0010 UART2 internal clock source = PLB 0011 Reserved	
4:7		Reserved	
8	U2EC	UART 2 EXT Clock Enable 0 UART2 uses the internal serial clock 1 UART2 uses the external serial clock	
9	U2DTE	UART 2 DMA Transmit Enable 0 UART2 DMA transmit is disabled 1 UART2 DMA transmit is enabled	
10	U2DRE	UART 2 DMA Receive Enable 0 UART2 DMA receive is disabled 1 UART2 DMA receive is enabled	
11	U2DC	UART 2 DMA Clear 0 U2DTE and U2DRE are not cleared when UART receives a corresponding terminal count. 1 U2DTE and U2DRE are cleared when UART receives a corresponding terminal count.	
12:23		Reserved	
24:31	U2DIV	UART 2 Divider 0000_0000 - UART2 divisor = 256 0000_0001 - UART2 divisor = 1 0000_0010 - UART2 divisor = 2 ..... 1111_1111 - UART2 divisor = 255	The source for SDR0_UART2[U2DIV] is indicated by SDR0_UART2[U2ICS]

31.4.4 UART Configuration Register 3 (SDR0_UART3)

Reset value = 0x20000026

Figure 31-16. UART Configuration Register 3 (SDR0_UART3)

0:3	U3ICS	UART 3 Internal Clock Source 0000 Reserved 0001 Reserved 0010 UART3 internal clock source = PLB 0011 Reserved	
4:7		Reserved	
8	U3EC	UART 3 EXT Clock Enable 0 UART3 uses the internal serial clock 1 UART3 uses the external serial clock	
9	U3DTE	UART 3 DMA Transmit Enable 0 UART3 DMA transmit is disabled 1 UART3 DMA transmit is enabled	
10	U3DRE	UART 3 DMA Receive Enable 0 UART3 DMA receive is disabled 1 UART3 DMA receive is enabled	
11	U3DC	UART 3 DMA Clear 0 U3DTE and U3DRE are not cleared when UART receives a corresponding terminal count. 1 U3DTE and U3DRE are cleared when UART receives a corresponding terminal count.	
12:23		Reserved	
24:31	U3DIV	UART 3 Divider 0000_0000 - UART3 divisor = 256 0000_0001 - UART3 divisor = 1 0000_0010 - UART3 divisor = 2 ..... 1111_1111 - UART3 divisor = 255	The source for SDR0_UART3[U3DIV] is indicated by SDR0_UART3[U3ICS]

31.5 FIFO Operation

This section discusses the various modes of operation, including interrupt mode, polled mode, and sleep mode.

31.5.1 Interrupt Mode

The following sections describe the operation of the UART in interrupt mode.

31.5.1.1 Receiver

Receiver interrupts occur as described below when the receiver FIFO and receiver interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ERBFI] = 1.

The received data available interrupt is issued when the number of characters in the FIFO has reached the trigger level programmed into UARTx_FCR. This interrupt is reset to 0 when the FIFO character count drops below this trigger level.

The received data available indicator is issued when the number of characters in the FIFO has reached the trigger level programmed into UARTx_FCR. This indicator is reset to 0 when the FIFO character count drops below this trigger level.

The receiver line status interrupt (UARTx_IIR = 0xC6) is a top priority interrupt, whereas the received data available interrupt (UARTx_IIR = 0xC4) is a second priority interrupt.

Data Ready (UARTx_LSR[DR]) is set as soon as a character is transferred from the shift register to the receiver FIFO. This bit is reset when the FIFO is empty.

Receiver timeout interrupts will occur as described below when the receiver FIFO and receiver interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ERBFI] = 1.

A FIFO timeout will occur when:

At least one character is in the receiver FIFO, no serial characters have been received for four serial character time periods, and the processor has not read the FIFO for four serial character time periods. A serial character time period is as follows:

1/(baud rate) x (# start bits + word length + # parity bits + # stop bits)

For example, the serial character time period for an 8-bit word with one parity bit, two stop bits at 56K baud is as follows:

1/(56000) x (1 + 8 + 1 + 2) = 214.3μs

So the timeout would occur after 857.1μs, if the above conditions hold.

When a timeout interrupt has occurred, it is cleared and the timer is reset when the processor reads one character from the receiver FIFO.

When a timeout interrupt has not occurred, the timer is reset after a new serial character is received or the processor reads the receiver FIFO.

31.5.1.2 Transmitter

Transmitter interrupts occur, as described below, when the transmitter FIFO and transmitter interrupts are enabled by setting UARTx_FCR[FE] = 1 and UARTx_IER[ETBEI] = 1.

The transmitter holding register interrupt (UARTx_IIR = 0xC2) occurs when transmit FIFO is empty, and is cleared as soon as the transmitter holding register is written to or the IIR is read. One to 16 characters may be written to the transmitter FIFO while servicing this interrupt.

The transmitter FIFO empty indications are delayed by one character time minus the last stop bit time whenever the following event occurs: UARTx_LSR[THRE] = 1 and there were less than two bytes simultaneously present in the transmit FIFO since the last UARTx_LSR[THRE] = 1. If UARTx_FCR[FE] = 1 (FIFOs enabled), the first transmitter interrupt after changing UARTx_FCR[FE] is immediate.

Receiver FIFO trigger level interrupts, received data available interrupts, and character timeouts all have equivalent second interrupt priority. Current transmitter holding register empty interrupt and Transmit FIFO empty have equivalent third interrupt priority.

31.5.2 Polled Mode

When UARTx_FCR[FE] = 1 (FIFOs enabled), and UARTx_IER[5:7] are all set to 0 (interrupts disabled), the UART is in FIFO polled mode of operation. The receiver and transmitter are controlled separately, so either can be in polled mode of operation. In polled mode, the user program must check the UARTx_LSR to see the status of the receiver and/or transmitter.

UARTx_LSR[3:6] specifies which errors (if any) have occurred. Character status errors are handled in the same way as in interrupt mode. Since UARTx_IER[ELSI] = 0, the IIR is not affected. UARTx_LSR[DR] is set as long as there is at least one character in the receiver FIFO. UARTx_LSR[THRE] indicates if the transmitter FIFO is empty. UARTx_LSR[TEMT] indicates if the transmitter FIFO and the transmitter shift register are empty. UARTx_LSR[RFE] indicates if there are any errors in the receiver FIFO.

In FIFO polled mode, there are no character timeout or trigger levels; however, the FIFOs are still capable of holding characters.

31.5.3 UART and Sleep Mode

All UARTs can be placed in sleep mode via the UART sleep bits in the CPM0_ER register (CPM0_ER[UART0:UART1:UART2_UART3]). The most common usage would be to save a little power if one or any of the UARTs were not going to be used.

Using sleep mode dynamically requires careful software control to make sure the UARTs are idle before putting them to sleep.

31.6 DMA Operation

The DMA controller can be configured to perform DMA operations using UART, which appears as an 8-bit peripheral to the DMA controller. When selected, the UART receiver is internally wired to the DMAReq and DMAAck signals of DMA channel 2, and the transmitter is internally wired to the DMAReq and DMAAck signals of DMA channel 3.

The UART can be operated in FIFO mode or non-FIFO mode. In FIFO mode, the transfers can be done as single transfers (DMA mode 0) or multiple transfers (DMA mode 1), depending on the setting of the DMS field in the FIFO Control Register (FCR). In non-FIFO mode, DMA transfers are performed using single transfers, using the UART's DMA mode 0. This section describes proper UART DMA programming.

DMA2P40 is attached to the 64-bit processor local bus. *Table 31-6* lists the DMA to PLB4 channel assignments in the PPC460EX/EXr/GT.

Table 31-6. DMA to PLB4 Channel Assignments

Channel	Sub-Channel	DMA Internal Source	DMA External Source
Channel 0	Sub-Channel 0	UART 0 Receive	External Peripheral DMA 0
	Sub-Channel 1	UART 2 Receive	
	Sub-Channel 2		
	Sub-Channel 3		
	Sub-Channel 4		
	Sub-Channel 5		
	Sub-Channel 6		
	Sub-Channel 7		
Channel 1	Sub-Channel 0	UART 0 Transmit	External Peripheral DMA 1
	Sub-Channel 1	UART 2 Transmit	
	Sub-Channel 2		
	Sub-Channel 3		
	Sub-Channel 4		
	Sub-Channel 5		
	Sub-Channel 6		
	Sub-Channel 7		
Channel 2	Sub-Channel 0	UART 1 Receive	External Peripheral DMA 2
	Sub-Channel 1	UART 3 Receive	
	Sub-Channel 2		
	Sub-Channel 3		
	Sub-Channel 4		
	Sub-Channel 5		
	Sub-Channel 6		
	Sub-Channel 7		
Channel 3	Sub-Channel 0	UART 1 Transmit	External Peripheral DMA 3
	Sub-Channel 1	UART 3 Transmit	
	Sub-Channel 2		
	Sub-Channel 3		
	Sub-Channel 4		
	Sub-Channel 5		
	Sub-Channel 6		
	Sub-Channel 7		

31.6.1 Transmitter DMA Mode

The four DMA channels on the PPC460EX/EXr/GT can be used to move data to and from the four UARTs. The UART0 and UART2 transmit channels subchannels of DMA channel 1. UART1 and UART3 transmit channels are subchannels of DMA channel 3. Use of the DMA with the UARTs is controlled by the UxDTE bits of the SDR0_UART0:3 registers.

When the DMA mode bit in the UART FIFO control register, UARTx_FCR[DMS], is 0 the DMA request goes active when the FIFOs are disabled or the FIFOs are enabled and there are no characters in the TX FIFO or Transmit Holding Register (THR). Once activated, the DMA request goes inactive after the first character is loaded into the TX FIFO or THR.

When UARTx_FCR[DMS] is 1 the DMA request goes active, when FIFOs are enabled and there is at least one unfilled position in the TX FIFO. DMA request goes inactive when the TX FIFO is completely full. To operate in this mode the corresponding PPC460EX/EXr/GT DMA Channel Control Register must be configured to accept DMA requests from an internal source. Setting the Peripheral Location (PL) bit of DMA2P40_CRx to 1 configures the DMA channel to accept DMA requests from UARTs on the OPB.

Table 31-7 lists required register settings for UART 0 and 2 transmit transfers. Other DMA registers and register fields must be programmed appropriately.

Table 31-7. UART x Where x = 0 or 2 Transmitter DMA Mode Register Field Settings

Register [Field]	Meaning
SDR0_UARTx[U0DTE] = 1	UARTx DMA transmit channel is enabled using DMA channel 1
SDR0_UARTx[U0DC]	Set to 0 to not clear SDR0_UART0[DTE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA2P40_CR1[TD] = 0	DMA channel 1 transfer direction is from memory to peripheral.
DMA2P40_CR1[PL] = 1	DMA channel 1 peripheral is on the OPB (UARTx).
DMA2P40_CR1[PW] = 000	Peripheral width is byte (8 bits).
DMA2P40_CR1[TM] = 00	DMA Channel 1 is in peripheral mode.
DMA2P40_CR1[PWC] = 000010	Peripheral wait cycles, how long the internal DMAck is active. Three cycles are required.
DMA2P40_CR1[PHC] = 000	Peripheral hold cycles are 0.
DMA2P40_CR1[ETD] = 1	EOT/TC is programmed as terminal count output.
DMA2P40_SC1[SCID] = 000	DMA Channel 1, sub-channel 0 selected.
UARTx_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.
Note: When using DMA channel 1 for UARTx transmitter transfers, external DMA transfers cannot be performed on this channel.	

Table 31-8 lists required register settings for UART 1 and 3 transmit transfers. The UARTx transmit channel is tied to DMA channel 3. Other DMA registers and register fields must be programmed appropriately.

Table 31-8. UART x Where x = 1 or 3 Transmitter DMA Mode Register Field Settings

Register [Field]	Meaning
SDR0_UARTx[U1DTE] = 1	UART x DMA transmit channel is enabled using DMA channel 3.
SDR0_UARTx[U1DC]	Set to 0 to not clear SDR0_UARTx[DTE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA2P40_CR3[TD] = 0	DMA channel 3 transfer direction is from memory to peripheral.
DMA2P40_CR3[PL] = 1	DMA channel 3 peripheral is on the OPB (UART0).
DMA2P40_CR3[PW] = 000	Peripheral width is byte (8 bits).
DMA2P40_CR3[TM] = 00	DMA channel 3 is in peripheral mode.
DMA2P40_CR3[PWC] = 000010	Peripheral wait cycles, how long the internal DMAck is active. Three cycles are required.
DMA2P40_CR3[PHC] = 000	Peripheral hold cycles are 0.
DMA2P40_CR3[ETD] = 1	EOT/TC is programmed as terminal count output.
DMA2P40_SC3[SCID] = 000	DMA Channel 3 sub-channel 0 selected.
UARTx_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.
Note: When using DMA channel 3 for UART x transmitter transfers, external DMA transfers cannot be performed on this channel.	

31.6.2 Receiver DMA Mode

The four DMA channels on the PPC460EX/EXr/GT can be used to move data to and from the four UARTs. The UART0 and UART2 receive channels are subchannels of DMA channel 0. UART1 and UART3 receive channels are subchannels of DMA channel 2. Use of the DMA with the UARTs is controlled by the UxDRE bits of the SDR0_UART0:3 registers.

When the DMA mode bit in the UART FIFO control register, UARTx_FCR[DMS], is 0 the DMA request goes active when there is at least one character in the RX FIFO or Receive Buffer Register (RBR). Once activated, the DMA request goes inactive when there are no more characters in the FIFO or RBR.

When UARTx_FCR[DMS] is 1 the DMA request goes active when the FIFOs are enabled and the trigger level or the timeout has been reached. DMA request goes inactive when there are no more characters in the FIFO or RBR. To operate in this mode the corresponding PPC460EX/EXr/GT DMA Channel Control Register must be configured to accept DMA requests from an internal source. Setting the Peripheral Location (PL) bit of DMA2P40_CRx to 1configures the DMA channel to accept DMA requests from UARTs on the OPB.

Table 31-9 lists required register settings for UART 0 and 2 receive transfers. Other DMA registers and register fields must be programmed appropriately.

Table 31-9. UART x Where x = 0 or 2 Receiver DMA Mode Register Field Settings

Register [Field]	Meaning
SDR0_UARTx[U0DRE] = 1	UARTx DMA receiver channel is enabled using DMA channel 0.
SDR0_UARTx[U0DC]	Set to 0 to not clear SDR0_UART0[DRE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA2P40_CR0[TD] = 1	DMA channel 0 transfer direction is from peripheral to memory.
DMA2P40_CR0[PL] = 1	DMA channel 0 peripheral is on the OPB (UARTx).
DMA2P40_CR0[PW] = 000	Peripheral width is byte (8 bits).
DMA2P40_CR0[TM] = 00	DMA channel 0 is in peripheral mode.
DMA2P40_CR0[PWC] = 000010	Peripheral wait cycles, how long the internal DMAAck is active. Three cycles are required.
DMA2P40_CR0[PHC] = 000	Peripheral hold cycles are 0.
DMA2P40_CR0[ETD] = 1	EOT/TC is programmed as terminal count output.
DMA2P40_SC0[SCID] = 000	DMA Channel 0 sub-channel 0 selected.
UARTx_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.
Note: When using DMA channel 0 for UARTx receiver transfers, external DMA transfers cannot be performed on this channel.	

Table 31-10 lists required register settings for UART1and 3 receive transfers. Other DMA registers and register fields must be programmed appropriately.

Table 31-10. UART x Where x = 1 or 3 Receiver DMA Mode Register Field Settings

Register [Field]	Meaning
SDR0_UARTx[U1DRE] = 1	UARTx DMA receiver channel is enabled using DMA channel 2.
SDR0_UARTx[UIDC]	Set to 0 to not clear SDR0_UART1[DRE] enable when terminal count is reached, set to 1 to clear enable when terminal count is reached.
DMA2P40_CR2[TD] = 1	DMA Channel 2 transfer direction is from peripheral to memory.
DMA2P40_CR2[PL] = 1	DMA Channel 2 peripheral is on the OPB (UART0).
DMA2P40_CR2[PW] = 00	Peripheral width is byte (8 bits).
DMA2P40_CR2[TM] = 00	DMA Channel 2 is in peripheral mode.
DMA2P40_CR2[PWC] = 000010	Peripheral wait cycles, how long the internal DMAAck is active. Three cycles are required.
DMA2P40_CR2[PHC] = 000	Peripheral hold cycles are 0.
DMA2P40_CR2[ETD] = 1	EOT/TC is programmed as terminal count output.
DMA2P40_SC2[SCID] = 000	DMA Channel 2 sub-channel 0 selected.
UARTx_FCR[DMS]	Set to 0 for a single DMA transfer or 1 for multiple DMA transfers.
Note: When using DMA channel 2 for UARTx receiver transfers, external DMA transfers cannot be performed on this channel.	

32. Serial RapidIO (SRIO) (PPC460GT only)

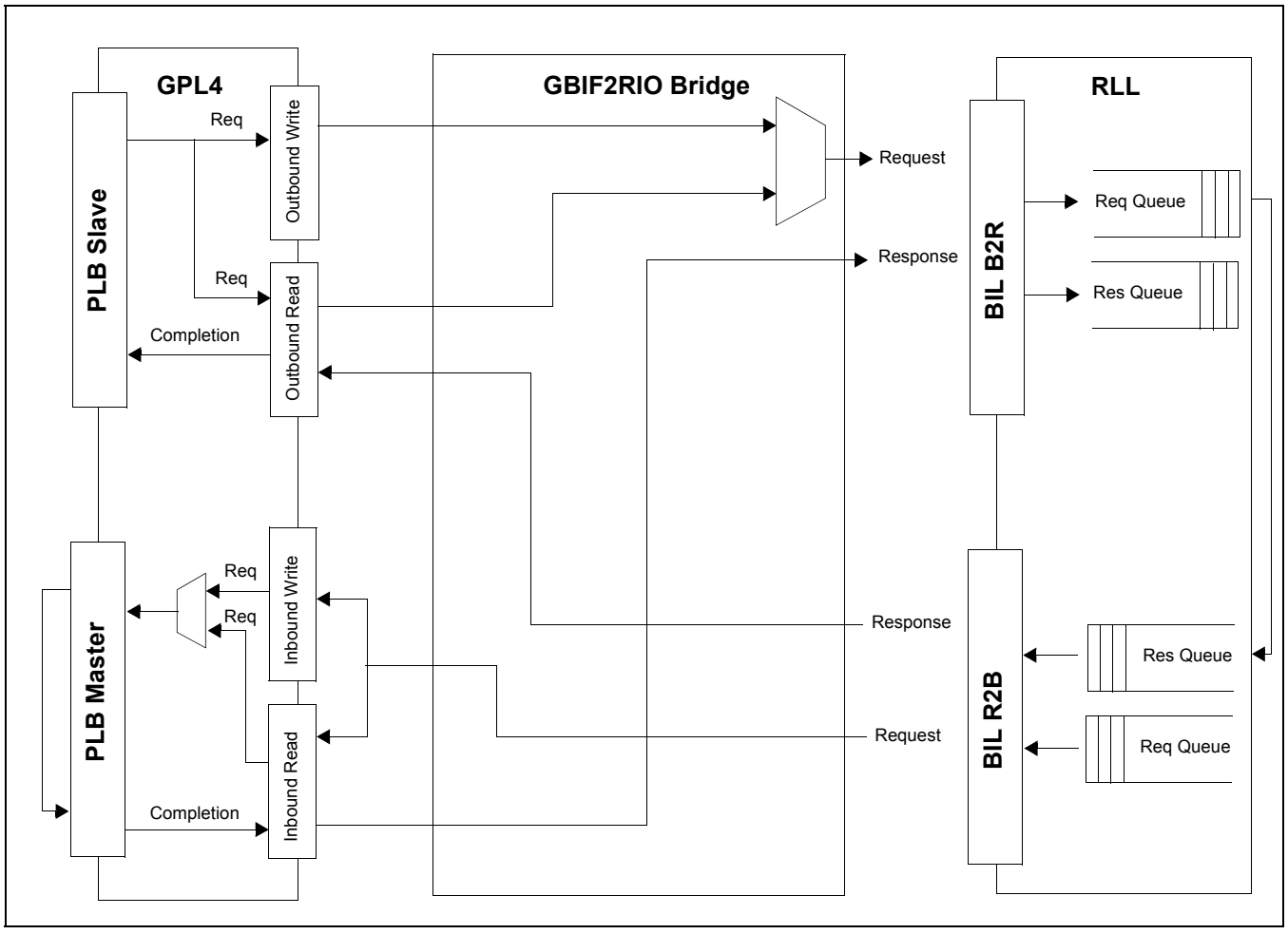
This section describes the operation of Serial RapidIO (SRIO) in the PPC460GT. This interface is available only on the PPC460GT.

The SRIO interface provides an interface to physical storage devices. It shares the High-Speed Serdes with the 4-lane PCI-Express interface.

Features include:

- Compliant with RapidIO Interconnect Specification, Revision 1.2
- Supports operation at 1.25 Gb/s, 2.50 Gb/s, and 3.125 Gb/s
- 1X/4X LP-SERIAL implementation of SRIO (except GSM extensions)
- SRIO Multicast Event (TOD) reception and transmission
- SRIO Error Management Extensions and Software Error Recovery
- Interoperability class 3 (complex mastering device)
- Device hot insertion/extraction
- Separate 4 KB transmit and receive request/response queues

Figure 32-1. SRIO Architecture



The SRIO block consists of three elements. The GPL4 block provides connection to the PLB4 internal bus. The RapidIO Logic Layer (RLL) connects to a 4-lane PHY. The GBIF2RIO Bridge connects these two elements.

32.1 Overview

SRIO provides a RapidIO based Endpoint which is interconnected by means of the on-chip PLB bus. The interface can be configured as one to four lanes. The interface shares a High-Speed Serdes with the four-lane PCI Express interface. The on-chip connection enables communication with other devices on the PLB, for example, the SDRAM memory controller and the PPC440 processor.

32.2 Functional Description

The SRIO implementation supports the following features.

- RapidIO CAR/CSR implementation
- Endpoint-Bridge CAR/CSR scalable implementation
- RapidIO and Backside Maintenance transaction handling
- RapidIO Maintenance PortWrite support
- RapidIO Mailbox transaction support for up to four mailboxes
- RapidIO Doorbell transaction support
- 34-bit bi-directional addressing support
- Support for RapidIO Error Management Extensions
- RapidIO Type 7 flow-control packets, passed to or from user interface
- Support for re-mapping RLL CAR/CSR space into inbound RapidIO memory map

32.2.1 SRIO Operating Frequency Limitations

There are two limitations to the PLB frequency as determined by the RLL frequency. (The RLL frequency is the line rate frequency divided by 20.)

- Minimum PLB frequency must be equal to or higher than the RLL frequency.
- Maximum PLB frequency must be less than 1.9 times the RLL frequency.

Table 32-1. SRIO Frequencies

Description	Rate	
Line Rate (Gbps)	2.5	3.125
RLL frequency (MHz)	125	156.25
PLB frequency (MHz)	125 - 245	156.25 – 300

32.2.2 SRIO Features not Supported

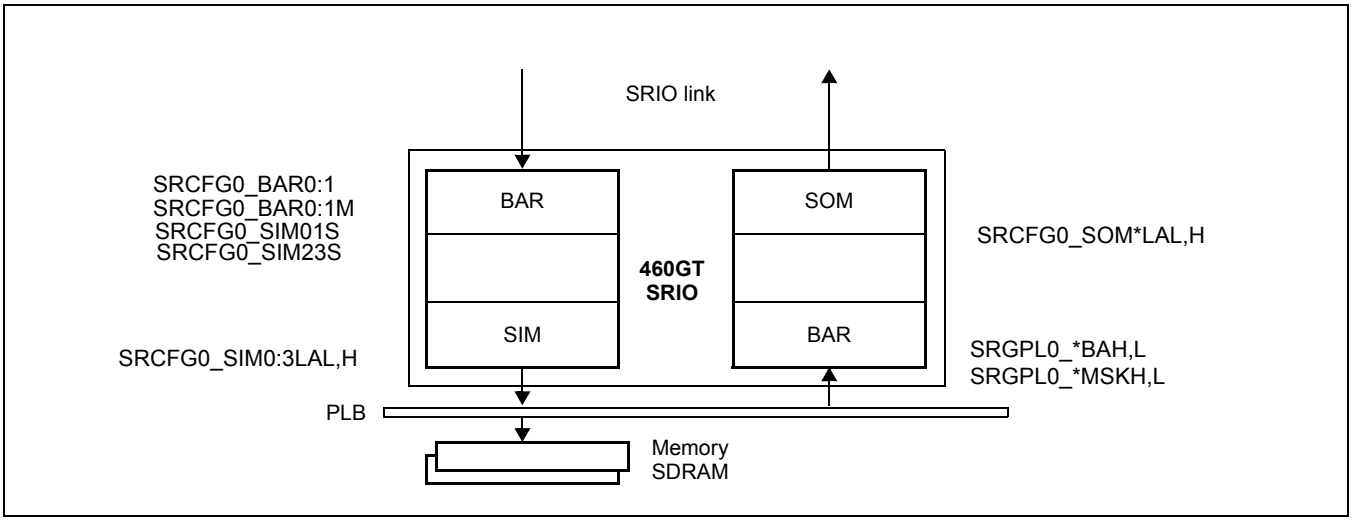
Long Run Transmitter AC timing in accordance with *RapidIO Interconnect Specification Part 6: 1X/4X LP-Serial Physical Layer Specification, Revision 1.3, 06/2005* are not supported. Only Short Run Transmitter AC timing specifications are supported.

32.3 Address Translation

The SRIO port supports several address ranges, also called regions, in both PLB and SRIO address spaces.

The registers involved in address translation for both inbound and outbound transactions for SRIO are shown in *Figure 32-2* for memory access from a remote device.

Figure 32-2.Root and Endpoint Address Translation for Memory Access



32.3.1 Outbound Address Mapping

The SRIO PLB Slave interface supports the following independent address regions on the PLB:

- Memory region 1 for outbound memory requests—a wide memory area allocation up to 2^{63} bytes. It is only enabled if SRGPL0_OMR1MSKL[UOT] is set. Commands accessing this region are translated to Memory transactions on the SRIO fabric (writes with response only when BAR set to do so, bit 21 of Outbound Memory Region #1 Mask Low Register is set (1))
- Memory region 2 for outbound memory requests. Commands accessing this region are translated to Memory transactions (writes with response only when BAR set as to do so, bit 30 of Outbound Memory Region #2 Mask Low Register is set (1))
- Memory region 3 for outbound memory requests. Commands accessing this region are translated to Memory transactions (writes with response only when BAR set to do so, bit 30 of Outbound Memory Region #3 Mask Low Register is set (1)).
- Message region for message transactions. Commands accessing this region are translated to Mailbox transactions.
- Configuration region for configuration requests. Commands accessing this region are translated to Configuration transactions. GPL4 supports only PLB single beat Rd/Wr on Configuration operations. Configuration commands are used to access the CAR/CSR registers in the RLL, and additional expanded CAR/CSR registers in the GBIF2RIO Bridge.
- Register region for local registers (SRREG0_xxxx).
- Maintenance region for external SRIO device configuration accesses. Commands accessing this region are translated to maintenance transactions (writes with response only when response is enabled, bit 30 of Outbound Maintenance Region Mask Low Register is set (1)). Maintenance BAR targets CAR/CSR registers of external SRIO devices.

The size of each of these regions is programmable as described in *Table 32-1*:

Table 32-2. Outbound Memory Ranges

Region	PLB Address (64 bits)	SRIO Address (34 bits)	Size (Bytes)	Address Boundary	Access
Memory #1	BAR0 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	SOM0 0_0000_0000 3_FFFF_FFFF	2^{27} (128MB) to 2^{63}	N × Size	R/W
Memory #2	BAR1 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	SOM1 0_0000_0000 3_FFFF_FFFF	2^{20} (1MB) to 2^{63}	N × Size	R/W
Memory #3	BAR2 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	SOM2 0_0000_0000 3_FFFF_FFFF	2^{7} (128B) to 2^{63}	N × Size	R/W
Message	BAR3 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	SOMMSG 0_0000_0000 3_FFFF_FFFF	2^{16} (64KB) to 2^{32} (4GB)	N × Size	W
Configuration	BAR4 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	SOMCFG 0_0000_0000 3_FFFF_FFFF	2^{29} (512MB) to 2^{32} (4GB)	N × Size	R/W
Register	BAR5 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	NA	2^{7} (128B) to 2^{15} (32KB)	N × Size	R/W
Maintenance	BAR6 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	POMMNT	2^{7} (128B) to 2^{15} (4GB)	N × Size	R/W

Notes:

1. All address regions are programmable in locations within the PLB address space at a granularity of their size. For example, if the size is 4GB, then it begins on a 4-GB boundary.
2. Each region can be disabled or enabled using the VAL field defined in the mask register set.
3. For maintenance, configuration, and register regions, only single-beat reads or writes are supported.

Typically, the configuration of these regions is changed only during the initialization process, not during chip operation. The registers that define these regions must only be changed during reset, or after reset *and* when none of the regions are enabled.

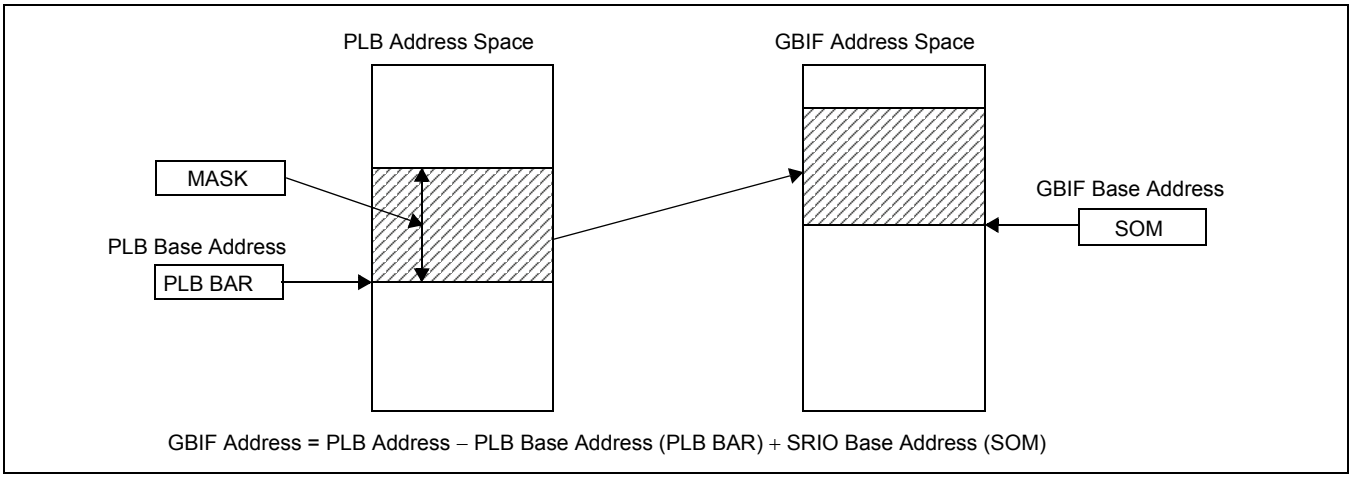
32.3.2 Outbound Address Translation

Each address region except the register region has an associated SRIO base address vector, known as the SRIO Outbound Map (SOM). The SOM maps a 64-bit PLB address to a 42-bit intermediate address. This intermediate address consists of the 8 bits of SRIO Device ID the command is targeting, and the 34-bit SRIO address. The PLB-to-SRIO address translation function is performed by concatenating the msbs of the SOM vector to the lsbs of the incoming PLB address. The number of msbs replaced is defined by the mask vector, which defines the size of the region. Only the msbs that are masked (set in the mask vector) are taken from the SOM vector. The remaining PLB address bits (lsbs) are unchanged.

The least significant 42 bits of the converted address (bits 22:63) are split into an 8-bit SRIO Device ID, and a 34-bit SRIO address based on the Device ID location specified in the SrioOutDestLoc register for the region accessed. The least significant three bits of the 34-bit address are always zero, so they are not presented to the SRIO function. The Device ID can be positioned in any location starting at any bit between 29:60. Care should be taken to ensure no single transaction crosses a GBIF address boundary that modifies the Device ID.

Figure 32-3 shows the outbound address translation logic. The PLB BAR vector defines a PLB address space offset. The SOM vector defines a SRIO address space offset plus 8 bits of Device ID. The size of both regions is identical and determined by the mask vector. The address translation function replaces the PLB space offset with the SRIO space offset by replacing the msbs from the PLB BAR vector with the msbs from the SOM vector, leaving all lsbs unchanged.

Figure 32-3.Outbound Address Translation Logic Step 1



For example, assume that the size of the memory region is 2^{57}. The mask vector is set so that bits 0:6 are ones and bits 7:63 are zeroes. The PLB Base Address (PLB BAR) of the region is set by PLB address bits 0:6. Thus, the SRIO address and SRIO Device ID is a concatenation of bits 0:6 from the SOM input to bits 7:63 from the PLB address, as shown in *Figure 32-4*.

Figure 32-4. Outbound Address Mapping Example Step 1 of 2

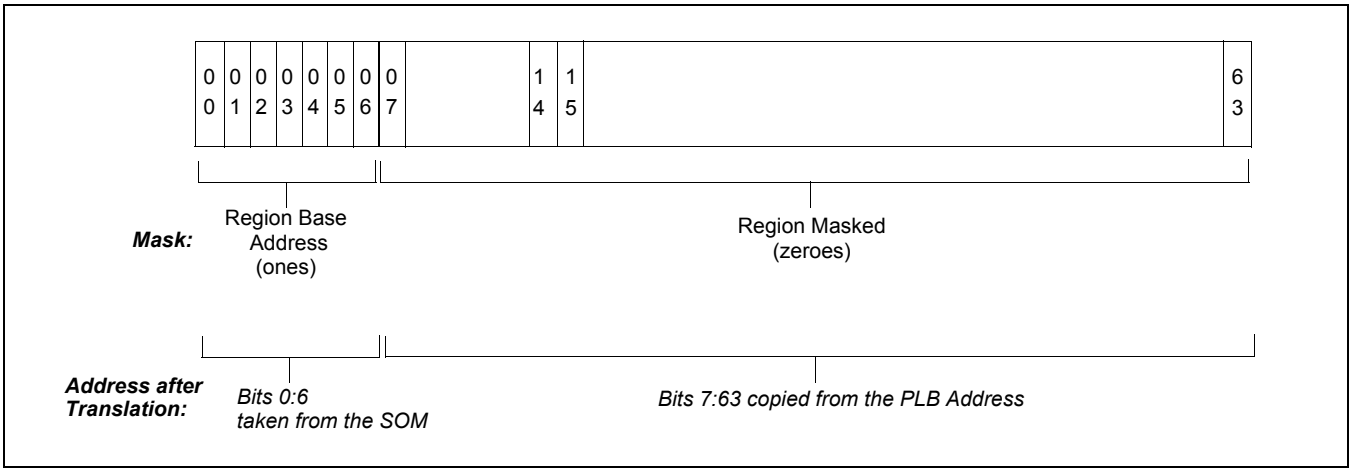


Figure 32-5. Outbound Address Mapping Example Step 2 of 2

GBIF Address[1]	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Ex1: Device ID Loc = 29																																							
DestID[7:0]	7	6	5	4	3	2	1	0																															
SRIO[33:3]									33	32	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
Ex2: Device ID Loc = 40																																							
DestID[7:0]												7	6	5	4	3	2	1	0																				
SRIO[33:3]	33	32	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
Ex3: Device ID Loc = 60																																							
DestID[7:0]																																7	6	5	4	3	2	1	0
SRIO[33:3]	33	32	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								
SRIO Addess = O_RIO_BASE+RIO_DEST_SIZE+#Destinations																																							
[1] GBIF Address[61:63] and GBIF Address[0:21] are used.																																							

The PLB Outbound BAR and mask registers are mapped within the DCR address space of the SRIO port as described in *Table 32-2*. The SOM registers are mapped within the Extended CAR/CSR address space.

Table 32-3. Outbound Region Registers

Region	DCR registers		CAR/CSR Registers
	PLB Bar	Mask	SOM/Destination ID
Memory #1	SRGPL0_OMR1BAH SRGPL0_OMR1BAL	SRGPL0_OMR1MSKH SRGPL0_OMR1MSKL	SRCCFG0_SOMOMR1LAH SRCFG0_SOMOMR1LAL SRCFG0_DIDLOMR1
Memory #2	SRGPL0_OMR2BAH SRGPL0_OMR2BAL	SRGPL0_OMR2MSKH SRGPL0_OMR2MSKL	SRCFG0_SOMOMR2LAH SRCFG0_SOMOMR2LAL SRCFG0_DIDLOMR2
Memory #3	SRGPL0_OMR3BAH SRGPL0_OMR3BAL	SRGPL0_OMR3MSKH SRGPL0_OMR3MSKL	SRCFG0_SOMOMR3LAH SRCFG0_SOMOMR3LAL SRCFG0_DIDLOMR3
Message	SRGPL0_MSGBAH SRGPL0_MSGBAL	SRGPL0_MSGMSK	SRCFG0_SOMMSGLAH SRCFG0_SOMMSGLAL SRCFG0_DIDLMSG
Configuration	SRGPL0_CFGBAH SRGPL0_CFGBAL	SRGPL0_CFGMSK	SRCFG0_SOMCFGLAH SRCFG0_SOMCFGLAL SRCFG0_DIDLCFG
Maintenance	SRGPL0_MNTBAH SRGPL0_MNTBAL	SRGPL0_MNTMSK	SRCFG0_POMMNTLAH SRCFG0_POMMNTLAL SRCFG0_DIDLMNT
Register	SRGPL0_REGBAH SRGPL0_REGBAL	SRGPL0_REGMSK	na

32.3.3 Inbound Address Mapping

The SRIO master interface supports up to two independent address regions in the SRIO address space.

- Memory region #0 for inbound memory requests. This region can be split into two non concatenated PLB memory regions. The SrioInSizeReg01 register determines the offset between these two regions within the SRIO address space.
- Memory region #1 for inbound memory requests. This region can be split into two non concatenated PLB memory regions. The SrioInSizeReg34 register determines the offset between these two regions within the SRIO address space.

The size of each of these regions is programmable as described in *Table 32-4* on page 1082.

Table 32-4. Inbound Memory Ranges

Region	SRIO Address (34 bits)	PLB Address (64 bits)	Size (Bytes)	Address Boundary	Access
Memory #0	BAR0 00_0000_0000 3_FFFF_FFFF	SIM0 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{10} (1KB) to 2^{63}	N × Size	R/W
		SIM1 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{20} (1MB) to 2^{63}		
Memory #1	BAR2 0_0000_0000 3_FFFF_FFFF	SIM3 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{10} (1KB) to 2^{63}	N × Size	R/W
		SIM4 0000_0000_0000_0000 FFFF_FFFF_FFFF_FFFF	2^{20} (1MB) to 2^{63}		

Notes:

1. All address regions are programmable in a location within the SRIO address space at a granularity of their size. For example, if the size is 4 GB, then it begins on a 4-GB boundary.
2. Each region can be disabled or enabled using the enable bit defined in the SrioInMaskRegn register in the SRIO configuration space.
3. No overlap is allowed between inbound address regions.

Typically, the configuration of these regions is only changed during the initialization process, not during chip operation. The registers that define those regions must only be changed during reset, or after reset *and* when none of the regions is enabled.

32.3.4 Inbound Address Translation

When an inbound transaction arrives, the PLB Master of the SRIO port compares the high msbs of the incoming SRIO address to each of the two SRIO BAR vectors. Only the bits that are masked (set to 1 in the mask vector) are compared against the incoming SRIO address. If a match occurs in one of the two regions, address translation is applied to the incoming SRIO address according to the selected region rules.

For address translation, each address region has at least one associated PLB Base Address vector, known as the SRIO Inbound Map (SIM). SRIO-to-PLB address translation is performed by concatenating msbs of the SIM vector to the lsbs of the incoming SRIO address. The number of msbs replaced is defined by the mask vector. Only the msbs that are masked (set in the mask vector) are taken from the SIM vector. The remaining SRIO address bits (lsbs) are unchanged. Using the SIM vector, the PLB Master interface claims a region within the PLB address space.

For example, assume that the size of the memory region is 1GB (2^{30}). The mask vector is set so that bits 0:34 are ones and bits 35:63 are zeroes. The offset address of the region is set by SRIO address bits 0:34, which should be equal to one of the two SRIO BARs. Thus, the PLB address is a concatenation of bits 0:34 from the SIM input to bits 35:63 from the SRIO address, as shown in *Figure 32-6*.

Figure 32-6.Inbound Address Mapping Example

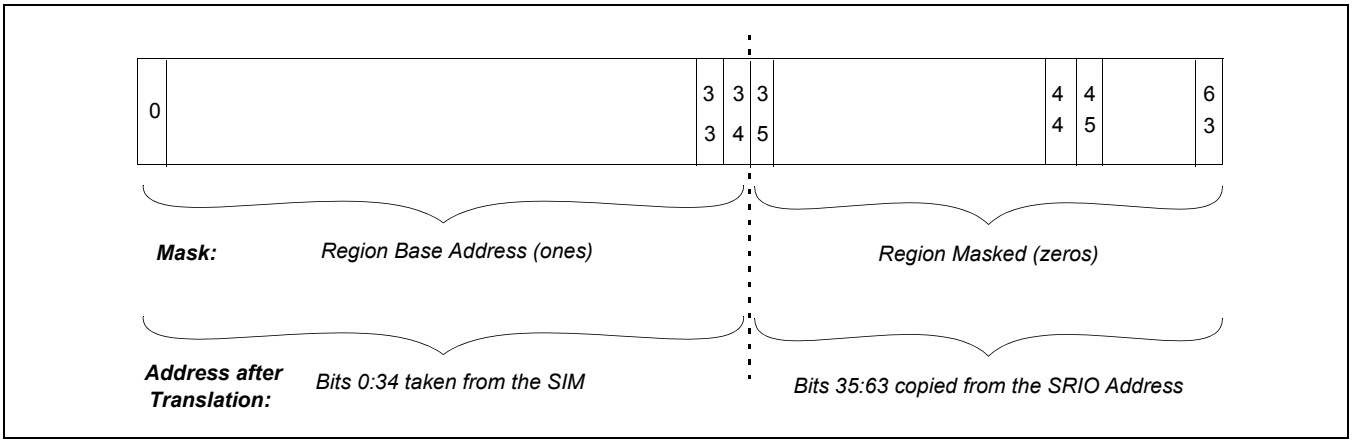
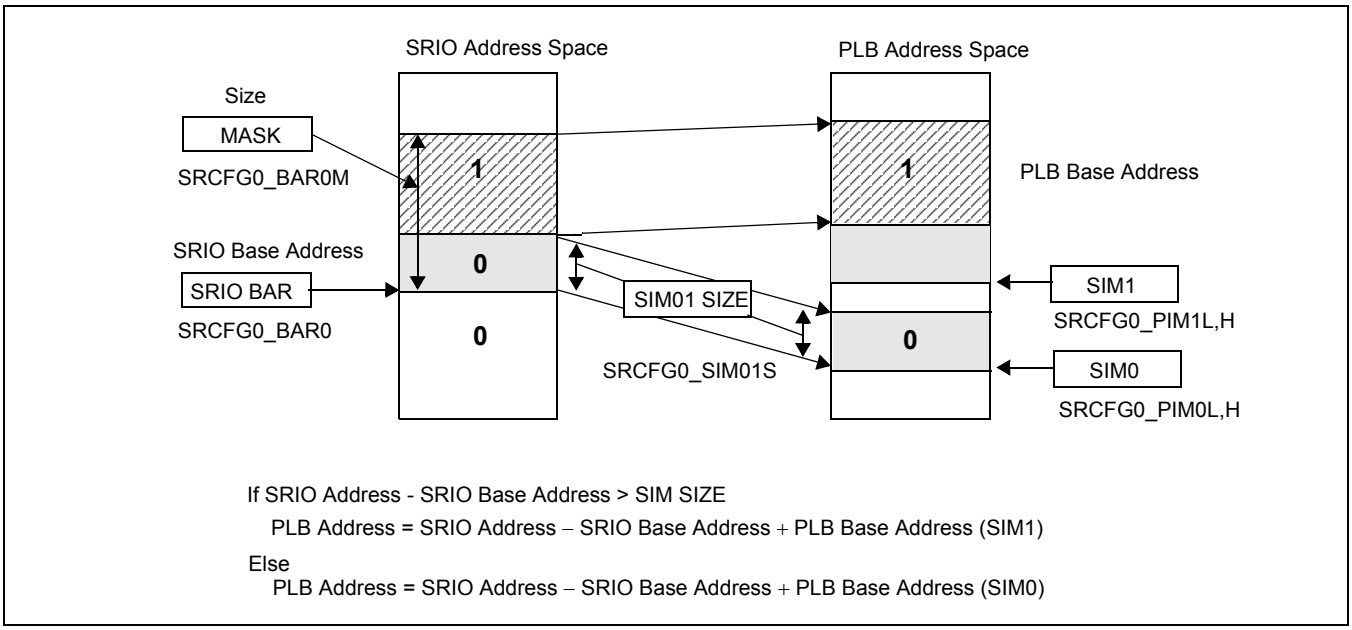


Figure 32-7 shows the Inbound Address Translation logic, using a double SIM. This logic applies to both regions. Before applying the address translation function described previously, the SRIO address is compared to an offset (SIM SIZE) that splits the address region into two regions.

Transactions located below the offset are translated into the PLB address space using the first SIM (SIM0). All other transactions are translated using the second SIM (SIM1). The value of the offset (SIM SIZE) is programmable and can range from 1KB to the size of the address region defined by the mask.

Figure 32-7. Inbound Address Translation with a Double SIM



The SRIO BAR, the mask, the SIM, and the SIM SIZE registers are mapped within the PLB configuration address space of the SRIO port, as shown in *Figure 32-4*.

Table 32-5. Inbound Region Registers

Address Space	PLB Space			
Region	**PCI BAR**	**MASK**	**PIM**	**PIM SIZE**
Memory #0	SRCFG0_BAR0	SRCFG0_BAR0M	SRCFG0_SIM0LAH SRCFG0_SIM0LAL	SRCFG0_SIM01S
			SRCFG0_SIM1LAH SRCFG0_SIM1LAL	
Memory #1	SRCFG0_BAR1	SRCFG0_BAR1M	SRCFG0_SIM2LAH SRCFG0_SIM2LAL	SRCFG0_SIM23S
			SRCFG0_SIM3LAH SRCFG0_SIM3LAL	

32.4 SRIO Registers

This section describes the SRIO registers in general. Included are the DCRs, SDRs, and application specific registers.

32.4.1 SRIO DCRs

The DCR Block is mapped into the DCR bus address space. Each SRIO port consists of 32 DCR addresses. The DCR base address for each SRIO port is defined as follows:

- SRIO port SR0: DCR base address is 0x140

These registers enable the controlling and monitoring of the SRIO PLB Slave interface, with the exception of the GPL Configuration Register (SRGPL0_CFG), which contains some bits that are related to the SRIO PLB Master interface.

Table 32-5 shows the DCR address offset mapping for the SRIO port.

Table 32-6. SRIO DCRs

Mnemonic	Description	Offset	Access	Page
SRGPL0_CFGBAH	Configuration Region Base Address High	0x00	R/W	1122
SRGPL0_CFGBAL	Configuration Region Base Address Low	0x01	R/W	1122
SRGPL0_CFGMSK	Configuration Region Mask and Validity	0x02	R/W	1122
SRGPL0_MSGBAH	Message Region Base Address High	0x03	R/W	1123
SRGPL0_MSGBAL	Message Region Base Address Low	0x04	R/W	1123
SRGPL0_MSGMSK	Message Region Mask and Validity	0x05	R/W	1124
SRGPL0_OMR1BAH	Outbound Memory Region 1 Base Address High	0x06	R/W	1124
SRGPL0_OMR1BAL	Outbound Memory Region 1 Base Address Low	0x07	R/W	1124
SRGPL0_OMR1MSKH	Outbound Memory Region 1 Mask High	0x08	R/W	1124
SRGPL0_OMR1MSKL	Outbound Memory Region 1 Mask and Validity Low	0x09	R/W	1125
SRGPL0_OMR2BAH	Outbound Memory Region 2 Base Address High	0x0A	R/W	1125
SRGPL0_OMR2BAL	Outbound Memory Region 2 Base Address Low	0x0B	R/W	1125
SRGPL0_OMR2MSKH	Outbound Memory Region 2 Mask High	0x0C	R/W	1126

Table 32-6. SRIO DCRs (Continued)

Mnemonic	Description	Offset	Access	Page
SRGPL0_OMR2MSKL	Outbound Memory Region 2 Mask and Validity Low	0x0D	R/W	1126
SRGPL0_OMR3BAH	Outbound Memory Region 3 Base Address High (BAR2H)	0x0E	R/W	1126
SRGPL0_OMR3BAL	Outbound Memory Region 3 Base Address Low (BAR2L)	0x0F	R/W	1126
SRGPL0_OMR3MSKH	Outbound Memory Region 3 Mask High	0x10	R/W	1127
SRGPL0_OMR3MSKL	Outbound Memory Region 3 Mask and Validity Low	0x11	R/W	1127
SRGPL0_REGBAH	Local Register Region Base Address High (BAR5H)	0x12	R/W	1127
SRGPL0_REGBAL	Local Register Region Base Address Low (BAR5L)	0x13	R/W	1128
SRGPL0_REGMSK	Local Register Region Mask and Validity	0x14	R/W	1128
SRGPL0_CFG	GPL Configuration Register	0x16	R/W	1128
SRGPL0_ESR	Error Status Register	0x17	R/W	1129
SRGPL0_EARH	Slave Error Address High Register	0x18	R	1129
SRGPL0_EARL	Slave Error Address Low Register	0x19	R	1129
SRGPL0_EATR	Slave Error Attribute Register	0x1A	R	1130
SRGPL0_MNTBAH	Maintenance Region Base Address High	0x1C	R/W	1130
SRGPL0_MNTBAL	Maintenance Region Base Address Low	0x1D	R/W	1130
SRGPL0_MNTMSK	Maintenance Region Mask	0x1E	R/W	1130

The reset value for all of the registers in the above table is 0.

The following sections describe the DCRs in detail.

32.4.2 Configuration Region Base Address High Register (SRGPL0_CFGBAH)

Functionally, the Configuration Region Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the Configuration Region (BAR4 for outbound non posted transactions).

Figure 32-8. Configuration Region Base Address High Register (SRGPL0_CFGBAH)

0:31	BAH	Configuration Region Base Address Bits (0:31)	

32.4.3 Configuration Region Base Address Low Register (SRGPL0_CFGBAL)

Figure 32-9. Configuration Region Base Address Low Register (SRGPL0_CFGBAL)

0:24	BAL	Configuration Region Base Address Bits (32:56): These bits can be masked to define an address space from 128B to 4GB in size.	
25:31		Reserved	

32.4.4 Configuration Region Mask and Validity Register (SRGPL0_CFGMSK)

The size of the Configuration Region is programmable from 128 bytes to 4 GB by means of a mask vector. This vector is used to mask out the unnecessary address bits in the address decoder. Bits 0:31 of the *Configuration Region Base Address High Register (SRGPL0_CFGBAH)* on page 1122 are always considered to be set, meaning

not masked. Bits 32:56 can be masked. Bits 56:63 are always considered to be "do not care" in the address decoding process, meaning masked. Setting a bit in the mask vector to 1 allows it to be taken into consideration in the address decoding process, meaning not masked. The mask vector must be programmed as a *leading ones* vector, indicating that if one of the bits is set, all the previous msbs must also be set.

Figure 32-10. Configuration Region Mask and Validity Register (SRGPL0_CFGMSK)

0:24	BAM	Configuration Region Base Address Mask: Bits 32:56 of the Configuration Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 128B (2^7) to 4GB (2^{32}): 00000000000000000000000000 - 2^{32}B 10000000000000000000000000 - 2^{31}B 11000000000000000000000000 - 2^{30}B --- 11111111111111111111111100 - 2^9B 11111111111111111111111110 - 2^8B 11111111111111111111111111 - 2^7B	
25:30		Reserved	
31	VAL	Specifies whether the Configuration Region is valid: 0 The SRIO PLB Slave interface does not respond to transactions that fall into this region.	

32.4.5 Message Region Base Address High Register (SRGPL0_MSGBAH)

Functionally, the Message Region Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the Message Region (BAR3 for outbound transactions).

Figure 32-11. Message Region Base Address High Register (SRGPL0_MSGBAH)

0:31	BAH	Message Region Base Address Bits (0:31)	

32.4.6 Message Region Base Address Low Register (SRGPL0_MSGBAL)

Figure 32-12. Message Region Base Address Low Register (SRGPL0_MSGBAL)

0:24	BAL	Message Region Base Address Bits 32:56: Bits 17:24 can be masked to define an address space from 128B to 32KB in size.	
25:31		Reserved	

32.4.7 Message Region Mask and Validity Register (SRGPL0_MSGMSK)

The size of the Message Region is programmable from 64 KB to 4 GB by means of a mask vector. For details about the mask vector, see *Configuration Region Mask and Validity Register (SRGPL0_CFGMSK)* on page 1122.

Figure 32-13. Message Region Mask and Validity Register (SRGPL0_MSGMSK)

0:15	BAM	Message Region Base Address Mask: Bits 32:47 of the Message Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 64 KB (2^{16}) to 4 GB (2^{32}).	
16:30		Reserved	
31	VAL	Specifies whether the Message Region is valid: 0 The SRIO PLB Slave interface does not respond to transactions that fall into this region.	

32.4.8 Outbound Memory Region 1 Base Address High Register (SRGPL0_OMR1BAH)

Functionally, the Outbound Memory Region 1 Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the first Outbound Memory Region (BAR0 for outbound transactions).

Figure 32-14. Outbound Memory Region 1 Base Address High Register (SRGPL0_OMR1BAH)

0:31	BAH	Outbound Memory Region 1 Base Address Bits 0:31	

32.4.9 Outbound Memory Region 1 Base Address Low Register (SRGPL0_OMR1BAL)

Figure 32-15. Outbound Memory Region 1 Base Address Low Register (SRGPL0_OMR1BAL)

0:4	BAL	Outbound Memory Region 1 Base Address Bits 32:36: BIts 1:36 can be masked to define an address space from 2^{27} (128MB) to 2^{63} in size.	
5:31		Reserved	

32.4.10 Outbound Memory Region 1 Mask High Register (SRGPL0_OMR1MSKH)

Functionally, the Outbound Memory Region 1 Mask High and Low Registers can be viewed as a single 64-bit register that defines the size of this address region. These registers must be programmed such that the concatenated mask is in the form of *leading ones*, meaning that if one of the bits is set, all the previous msbs must also be set. The size of the Outbound Memory Region 1 is programmable from 2^{27}(128 MB) to 2^{63} bytes by means of a mask vector.

Figure 32-16. Outbound Memory Region 1 Mask High Register (SRGPL0_OMR1MSKH)

0		Reserved	

1:31	BAMH	Outbound Memory Region 1 Mask Bits (1:31): Bits 1:36 of the Base Address can be masked. This region can range in size from 2^{27} (128MB) to 2^{63}. The mask defines the bits that are considered by the address decoder, thus determining the size of the region: 00000000000000000000000000000000000 - 2^{63}B 10000000000000000000000000000000000 - 2^{62}B 11000000000000000000000000000000000 - 2^{61}B --- 11111111111111111111111111111111100 - 2^{29}B 11111111111111111111111111111111110 - 2^{28}B 11111111111111111111111111111111111 - 2^{27}B	

32.4.11 Outbound Memory Region 1 Mask and Validity Low Register (SRGPL0_OMR1MSKL)

Figure 32-17. Outbound Memory Region 1 Mask and Validity Low Register (SRGPL0_OMR1MSKL

0:4	BAML	Outbound Memory Region 1 Mask Bits (32:36). See explanation for SRGPL0_OMR1MSKH[BAMH].	
5:20		Reserved	
21	WWR	Write With Response in the SRIO domain 0 Writes without response 1 Writes with response	
22:29		Reserved	
30	UOT	Use One TC: 1 Outbound Memory Region 1 is not divided into eight sub-regions, but is used as a single wide memory area.	Must be 1.
31	VAL	Specifies whether Outbound Memory Region 1 is valid: 0 The SRIO PLB Slave interface does not respond to transactions that fall into this region.	

32.4.12 Outbound Memory Region 2 Base Address High Register (SRGPL0_OMR2BAH)

Functionally, the Outbound Memory Region 2 Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the second Outbound Memory Region (BAR1 for outbound transactions).

Figure 32-18. Outbound Memory Region 2 Base Address High Register (SRGPL0_OMR2BAH)

0:31	BAH	Outbound Memory Region 2 Base Address Bits (0:31)	

32.4.13 Outbound Memory Region 2 Base Address Low Register (SRGPL0_OMR2BAL)

Figure 32-19. Outbound Memory Region 2 Base Address Low Register (SRGPL0_OMR2BAL)

0:11	BAL	Outbound Memory Region 2 Base Address Bits 32:43: Bits 1:43 can be masked to define an address space from 2^{20} (1 MB) to 2^{63} in size.	
12:31		Reserved	

32.4.14 Outbound Memory Region 2 Mask High Register (SRGPL0_OMR2MSKH)

Functionally, the Outbound Memory Region 2 Mask High and Low Registers can be viewed as a single 64-bit register that defines the size of this address region. The size of the Outbound Memory Region 2 is programmable from 2^{20} (1 MB) to 2^{63} bytes by means of a mask vector. For details about the mask vector, see *Outbound Memory Region 1 Mask High Register (SRGPL0_OMR1MSKH)* on page 1124.

Figure 32-20. Outbound Memory Region 2 Mask High Register (SRGPL0_OMR2MSKH)

0		Reserved	
1:31	BAMH	Outbound Memory Region 2 Mask Bits 1:31: Bits 1:43 of the Base Address can be masked to define an address space from 2^{20} (1MB) to 2^{63} bytes in size. The mask defines the bits that are considered by the address decoder, thus determining the size of the region.	

32.4.15 Outbound Memory Region 2 Mask and Validity Low Register (SRGPL0_OMR2MSKL)

Figure 32-21. Outbound Memory Region 2 Mask and Validity Low Register (SRGPL0_OMR2MSKL)

0:11	BAML	Outbound Memory Region 2 Mask Bits 32:43. See explanation for SRGPL0_OMR2MSKH[BAMH].	
12:29		Reserved	
30	WWR	Write With Response in the SRIO domain 0 Writes without response 1 Writes with response	
31	VAL	Specifies whether Memory Region 2 is valid: 0 The SRIO PLB Slave interface does not respond to transactions that fall into this region.	

32.4.16 Outbound Memory Region 3 Base Address High Register (SRGPL0_OMR3BAH)

Functionally, the Outbound Memory Region 3 Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the third Outbound Memory Region (BAR2 for outbound transactions).

Figure 32-22. Outbound Memory Region 3 Base Address High Register (SRGPL0_OMR3BAH)

0:31	BAH	Outbound Memory Region 3 base address bits 0:31	

32.4.17 Outbound Memory Region 3 Base Address Low Register (SRGPL0_OMR3BAL)

Figure 32-23. Outbound Memory Region 3 Base Address Low Register (SRGPL0_OMR3BAL)

0:24	BAL	Outbound Memory Region 3 base address bits 32:56: Bits 1:56 can be masked to define an address space from 2^{7} (128B) to 2^{63} in size.	

25:31		Reserved	

32.4.18 Outbound Memory Region 3 Mask High Register (SRGPL0_OMR3MSKH)

Functionally, the Outbound Memory Region 3 Mask High and Low Registers can be viewed as a single 64-bit register that defines the size of this address region. The size of the Outbound Memory Region 3 is programmable from 2^7 (128 bytes) to 2^{63} bytes by means of a mask vector. For details about the mask vector, see *Outbound Memory Region 1 Mask High Register (SRGPL0_OMR1MSKH)* on page 1124.

Figure 32-24. Outbound Memory Region 3 Mask High Register (SRGPL0_OMR3MSKH)

0		Reserved	
1:31	BAMH	Outbound Memory Region 3 Mask Bits 1:31: Bits 1:56 of the Base Address can be masked to define an address space from 2^7 (128B) to 2^{63} bytes in size. The mask defines the bits that are considered by the address decoder, thus determining the size of the region.	

32.4.19 Outbound Memory Region 3 Mask and Validity Low Register (SRGPL0_OMR3MSKL)

Figure 32-25. Outbound Memory Region 3 Mask and Validity Low Register (SRGPL0_OMR3MSKL)

0:24	BAML	Outbound Memory Region 3 Mask Bits 32:56. See explanation for SRGPL0_OMR3MSKH[BAMH].	
25:29		Reserved	
30	WWR	Write With Response in the SRIO domain 0 Writes without response 1 Writes with response	
31	VAL	Specifies whether Memory Region 3 is valid: 0 The SRIO PLB Slave interface does not respond to transactions that fall into this region.	

32.4.20 Local Register Region Base Address High Register (SRGPL0_REGBAH)

Functionally, the Local Register Region Base Address High and Low Registers can be viewed as a single 64-bit register that defines the PLB Base Address of the Local Register Region (BAR5).

Figure 32-26. Local Register Region Base Address High Register (SRGPL0_REGBAH)

0:31	BAH	Local Register Region Base Address Bits 0:31	

32.4.21 Local Register Region Base Address Low Register (SRGPL0_REGBAL)

Figure 32-27. Local Register Region Base Address Low Register (SRGPL0_REGBAL)			
0:24	BAL	Local Register Region Base Address Bits 32:56. Bits 1:56 can be masked to define an address space from 128B (2^7) to 32KB (2^{15}) in size.	
25:31		Reserved	

32.4.22 Local Register Region Mask and Validity Register (SRGPL0_REGMSK)

The size of the Local Register Region is programmable from 128 bytes to 32 KB, by means of a mask vector. For details about the mask vector, see *Configuration Region Mask and Validity Register (SRGPL0_CFGMSK)* on page 1122.

Figure 32-28. Local Register Region Mask and Validity Register (SRGPL0_REGMSK)			
0:16		Reserved	
17:24	BAM	Register Region Base Address Mask: Bits 49:56 of the Local Register Region Base Address can be masked. The mask defines the bits that are considered by the address decoder, thus determining the size of the region. This region can range in size from 128B (2^7) to 32KB (2^{15}): 00000000 - 32KB 10000000 - 16KB 11000000 - 8KB --- 11111111 - 128B	
25:30		Reserved	
31	VAL	Specifies whether the register region is valid: 0 The SRIO PLB Slave interface does not respond to transactions that fall into this region.	

32.4.23 GPL Configuration Register (SRGPL0_CFG)

Figure 32-29. GPL Configuration Register (SRGPL0_CFG)			
0:1		Reserved	
2	PLE	Inbound Read Pipeline Enable	
3	MGA	SRIO PLB Master Guarded Attribute	
4:5	MPRI	SRIO PLB Master Priority	
6	DMER	Disable Slave MERR Assertion: Setting this bit to 1 suppresses the error assertion (Read and Write Error) if a transaction error is detected by the SRIO PLB Slave interface.	
7	DTDC	Disable Target Directed Completion (TDC) Assertion: Setting this bit to 1 suppresses the TDC mechanism in the SRIO PLB Slave interface.	
8:31		Reserved	

32.4.24 Error Status Register (SRGPL0_ESR)

Bits 0:5 of this register are read only, write 1 to clear (R/W1C). Register bits indicate the status when read. A set bit when write indicates that the status event bit can be cleared. Writing a 0 has no effect.

Figure 32-30. Error Status Register (SRGPL0_ESR)

0	SEV	Slave Error Valid Indication: If asserted, SRGPL0_EARH, SRGPL0_EARL, and SRGPL0_EATR contain valid information.	
1	MIRQ	MIRQ Received Indication: MIRQ exception feedback from any PLB Slave to the SRIO PLB Master interface.	
2	MERR	MERR Received Indication: Transaction error (Read and Write) indication feedback from any PLB Slave during a transaction initiated by the SRIO PLB Master interface.	
3	TOT	Time-Out Received Indication: Time-Out indication feedback from the PLB Arbiter during a transaction initiated by the SRIO PLB Master interface.	
4	ORPE	Outbound Read Parity Error.	
5	IWPE	Inbound Write Parity Error.	
6	IE	Interrupt Indication Enable: If set to 1, the SRIO DCR interrupt is raised if any of the previous bits (0:5) are set.	
7:31		Reserved	

32.4.25 Slave Error Address High Register (SRGPL0_EARH)

Figure 32-31. Slave Error Address High Register (SRGPL0_EARH)

0:31	SEARH	Upper PLB Address provided to the SRIO PLB Slave interface if an error occurs	

32.4.26 Slave Error Address Low Register (SRGPL0_EARL)

Figure 32-32. Slave Error Address Low Register (SRGPL0_EARL)

0:31	SEARL	Lower PLB Address provided to the SRIO PLB Slave interface if an error occurs	

32.4.27 Slave Error Attribute Register (SRGPL0_EATR)

Figure 32-33. Slave Error Attribute Register (SRGPL0_EATR)

0	SERNW	Read Not Write Attribute of the PLB Master addressing the SRIO PLB Slave interface if an error occurs: 0 SRIO PLB Slave Write operation. 1 SRIO PLB Slave Read operation.	
1:4	SESIZE	Size Attribute of the PLB Master addressing the SRIO PLB Slave interface if an error occurs	
5:8	SEMID	ID Attribute of the PLB Master addressing the SRIO PLB Slave interface if an error occurs	
9:31		Reserved	

32.4.28 Maintenance Region Base Address High Register (SRGPL0_MNTBAH)

Figure 32-34. Maintenance Region Base Address High Register (SRGPL0_MNTBAH)

0:31	BAH	Maintenance region Base Address High	

32.4.29 Maintenance Region Base Address Low Register (SRGPL0_MNTBAL)

Figure 32-35. Maintenance Region Base Address Low Register (SRGPL0_MNTBAL)

0:24	BAL	Maintenance region Base Address Low	
25:31		Reserved	

32.4.30 Maintenance Region Mask Register (SRGPL0_MNTMSK)

Figure 32-36. Maintenance Region Mask Register (SRGPL0_MNTMSK)

0:24	BAM	Maintenance region base address mask bits 32:56 (out of 64)	
25:29		Reserved	
30	WWR	Write With Response in the SRIO domain 0 Maintenance writes without response 1 Maintenance writes with response	
31	VAL	Valid register region 0 GPLS does not respond to transactions which fall into this region 1 GPLS responds to transactions which fall into this region	

32.5 SRIO SDRs

Table 32-6 lists the System DCR registers used for the SRIO ports.

Reset values are field dependent and appear in the register figure Comment column.

Table 32-7. SRIO SDRs

Register	Description	Address	Access	Page
SDR0_SRIO_CFG	SRIO Configuration Register	0x0380	R/W	1131
SDR0_SRIO_PCSSSSTS	SRIO PCS Sync State Status Register	0x0381	R	1132
SDR0_SRIO_PCSSSTS	SRIO PCS Sync Status Register	0x0382	R	1133
SDR0_SRIO_PCSASTS	SRIO PCS Alignment Status Register	0x0383	R	1134
SDR0_SRIO_RLLSTS	SRIO RLL Status Register	0x0384	R	1135

The following sections describe the SDRs in detail.

32.5.1 SRIO Configuration Register (SDR0_SRIO_CFG)

Figure 32-37.SRIO Configuration Register (SDR0_SRIO_CFG)

0:12		Reserved	
13	RX_COMP_SYNC	Force Receive Compensation Synchronization Active-high pulse in core_clk domain to set the wptr-to-rptr distance in the clock compensation FIFO.	
14	RX_PH_SYNC	Force Receive Phase Synchronization Force the receive phase compensators to initialize the wptr-to-rptr distance in the alignment FIFO. Must be asserted for at least 2 rxclk_a cycles.	
15	TX_PH_SYNC	Force Transmit Phase Synchronization Active-high pulse in core_clk domain to set the wptr-to-rptr distance once the four tx_clks and core_clk are stable.	
16:24		Reserved	
25	EN_SRIO_PCS_SNRR	Enable SRIO PCS Serdes Not Ready Reset Enable the Serdes Not Ready condition to reset the SRIO PCS.	Reset value = 1 (enable).
26	EN_SRIO_PCS_PNLR	Enable SRIO PCS PLL Not Locked Reset Enable the PLL Not Locked condition to reset the SRIO PCS.	Reset value = 1 (enable).
27	EN_SRIO_PCS	Enable SRIO PCS Active high. It allows user to disable SRIO IP when not active.	Reset value = 1 (enable).
28	EN_PMA_SIGDET	Enable PMA Signal Detect When low, it activates internal logic that uses the transition from a SRIO |A| K27.7 to a non-|A| in two sequential recovered-clock cycles as proof that there is a valid signal driving the receive side of that lane. It allows user to force ignore if serdes sigdet does not work.	Reset value = 1 (enable).

29:31	SRIO_INB_CTL	SRIO Inband Control When low, it activates internal logic that uses the transition from a SRIO |A| K27.7 to a non-|A| in two sequential recovered-clock cycles as proof that there is a valid signal driving the receive side of that lane. It allows user to force ignore if serdes sigdet does not work.	Reset value = 0b111 (enable).

32.5.2 SRIO PCS Sync State Status Register (SDR0_SRIO_PCSSSSTS)

Figure 32-38.SRIO PCS Sync State Status Register (SDR0_SRIO_PCSSSSTS)

0:7	SYNC_STATE_L3	Sync State for Lane 3 Current state of the SRIO Lane_Sync state machine for Lane 3. [Ref: SRIO Specifications Part 6 SRIO PHY]. The one-hot encodings are as follows (this assumes that the synthesis operation does not recode these states): 0000_0001: NO_SYNC 0000_0010: NO_SYNC_1 0000_0100: NO_SYNC_2 0000_1000: SYNC 0001_0000: SYNC_1 0010_0000: SYNC_2 0100_0000: SYNC_3 1000_0000: SYNC_4 Others: *Illegal*, transitions to NO_SYNC next cycle.	Synchronous to rxadclk.
8:15	SYNC_STATE_L2	Sync State for Lane 2 Current state of the SRIO Lane_Sync state machine for Lane 2.	
16:23	SYNC_STATE_L1	Sync State for Lane 1 Current state of the SRIO Lane_Sync state machine for Lane 1.	
24:31	SYNC_STATE_L0	Sync State for Lane 0 Current state of the SRIO Lane_Sync state machine for Lane 0.	

32.5.3 SRIO PCS Sync Status Register (SDR0_SRIO_PCSSSTS

Figure 32-39. SRIO PCS Sync Status Register (SDR0_SRIO_PCSSSTS)			
0:15		Reserved	
16	L3_SYNC	Lane 3 is in SRIO Sync	
17	L2_SYNC	Lane 2 is in SRIO Sync	
18	L1_SYNC	Lane 1 is in SRIO Sync	
19	L0_SYNC	Lane 0 is in SRIO Sync	
20	SYNC_VIOLATE_L3	Sync Violation for Lane 3 The 10b8b decoder for Lane 3 has detected an error in one or both of the 10b code-groups decoded in the current cycle. This signal is synchronous to rxadclk. This signal may be connected by the user to a counter circuit, monitored during the calibration of the SRIO link (i.e. while adjusting RX equalization and other serdes parameters to accommodate link characteristics).	
21	SYNC_VIOLATE_L2	Sync Violation for Lane 2 The 10b8b decoder for Lane 2 has detected an error in one or both of the 10b code-groups decoded in the current cycle.	
22	SYNC_VIOLATE_L1	Sync Violation for Lane 1 The 10b8b decoder for Lane 1 has detected an error in one or both of the 10b code-groups decoded in the current cycle.	
23	SYNC_VIOLATE_L0	Sync Violation for Lane 0 The 10b8b decoder for Lane 0 has detected an error in one or both of the 10b code-groups decoded in the current cycle.	
24	SYNC_DETECT_L3	Sync Detect for Lane 3 The receive PCS logic has detected one or two "commas" (K28.5) on Lane 3 in the current cycle. This signal is synchronous to rxadclk. This signal may be connected by the user to a counter circuit, monitored during the calibration of the SRIO link (i.e. while adjusting RX equalization and other serdes parameters to accommodate link characteristics).	
25	SYNC_DETECT_L2	Sync Detect for Lane 2 Receive PCS logic has detected one or two "commas" (K28.5) on Lane 2 in the current cycle.	
26	SYNC_DETECT_L1	Sync Detect for Lane 1 The receive PCS logic has detected one or two "commas" (K28.5) on Lane 1 in the current cycle.	
27	SYNC_DETECT_L0	Sync Detect for Lane 0 The receive PCS logic has detected one or two "commas" (K28.5) on Lane 0 in the current cycle.	

28	SIGNAL_DETECT_L3	Signal Detect for Lane 3 Reflects the value of the external High-Speed SerDes L3_RX_SIGDET pin when enable_pma_sigdet = 0b1, else it reflects the output of the SRIO PCS internal signal detect circuit.	This output is considered Asynchronous.
29	SIGNAL_DETECT_L2	Signal Detect for Lane 2 Reflects the value of the external High-Speed SerDes L2_RX_SIGDET pin when enable_pma_sigdet = 0b1, else it reflects the output of the SRIO PCS internal signal detect circuit.	This output is considered Asynchronous.
30	SIGNAL_DETECT_L1	Signal Detect for Lane 1 Reflects the value of the external High-Speed SerDes L1_RX_SIGDET pin when enable_pma_sigdet = 0b1, else it reflects the output of the SRIO PCS internal signal detect circuit.	This output is considered Asynchronous.
31	SIGNAL_DETECT_L0	Signal Detect for Lane 0 Reflects the value of the external High-Speed SerDes L0_RX_SIGDET pin when enable_pma_sigdet = 0b1, else it reflects the output of the SRIO PCS internal signal detect circuit.	This output is considered Asynchronous.

32.5.4 SRIO PCS Alignment Status Register (SDR0_SRIO_PCSASTS)

Figure 32-40. SRIO PCS Alignment Status Register (SDR0_SRIO_PCSASTS)

0	ALIGNED	Lanes 0-3 Aligned All four lanes of the SRIO have achieved alignment.	
1:15		Reserved	
16:23	ALIGNED_VEC	Aligned Vector Indicates the presence of align characters [K27.7] in any of the 8 bytes of the 64-bit receive data bus on a per-cycle basis. Valid encodings are as follows: 0000_0000: No align characters present 1111_0000: Aligned Column Received 0000_1111: Aligned Column Received Others: Invalid, Align Error Present Link tries to recover.	Synchronous to core_clk.
24	ALIGNED_ERROR	Aligned Error Asserts for every cycle where an align character [K27.7] is detected on at least one of the four lanes, but not simultaneously on all four lanes.	This signal has meaning only when operating the SRIO link in 4X mode. Synchronous to core_clk.

25:31	ALIGNED_STATE	Aligned State Current state of the SRIO Lane_Align state machine. The one-hot encodings are as follows (this assumes that the synthesis operation does not recode these states): 000_0001: NOT_ALIGNED 000_0010: NOT_ALIGNED_1 000_0100: NOT_ALIGNED_2 000_1000: ALIGNED 001_0000: ALIGNED_1 010_0000: ALIGNED_2 100_0000: ALIGNED_3 Others: ILLEGAL, transition sto NO_SYNC next cycle.	Synchronous to core_clk.

32.5.5 SRIO RLL Status Register (SDR0_SRIO_RLLSTS)

Figure 32-41. SRIO RLL Status Register (SDR0_SRIO_RLLSTS)

0:25		Reserved	
26	ECRC_GOOD	ECRC Good Status Provides early indication of a mismatch of a packet's ECRC field.	
27	TX_CON_ACK	Transmit Control Symbol Acknowledgment R2_ip_ser3 acknowledgement of the transmission of a user-requested SRIO control symbol.	
28		Reserved	
29:31	COMP_STAT	Clock Compensation Status How close the SRIO transmitter is to forcing the transmission of a clock compensation sequence.	

32.6 SRIO Application Specific Registers

The Application Specific Register block contains registers that are used for application specific parameter configuration and status.

32.6.1 Capability Registers (CARs) and Command and Status Registers (CSRs) Abbreviations

The following table provides a list of all the abbreviations found in the CARs and CSRs used in the SRIO function.

Table 32-8. CAR/CSR Bit Control Notation Abbreviations

Abbreviation	Description
R/W	Read/Write (From Maint or CFG op)
R	Read Only (From Maint or CFG op)
R/Clear	Readable status bit/Write 1 to Clear (From Maintenance or CFG op)
0	Register value is initialized to 0.
1	Register value is initialized to 1.
Reserved	Do not write. (Writes ignored, read back undefined.)

32.6.2 SRIO Configuration Registers (CAR/CSR)

The following table provides a list of all the capability registers (CARs) and the configuration and status registers (CSRs) used by the SRIO function.

Note: Writes to addresses above 0x12000 that are not defined in the following table cause unexpected results. Other registers in this region can be corrupted.

The PLB base address to access these registers is defined by the dedicated DCR registers SRGPL0_CFGBAH/L, and SRGPL0_CFGMSK.

Table 32-9. SRIO CARs/CSRs

Mnemonic	Description	Offset	Access	Page
SRIO CAR/CSR Registers				
SRCFG0_DEVID	Device ID CAR	0x0000	R	1138
SRCFG0_DEVINFO	Device Information CAR	0x0004	R	1139
SRCFG0_ASSEMBID	Assembly ID CAR	0x0008	R	1139
SRCFG0_ASEMBINFO	Assembly Information CAR	0x000C	R	1139
SRCFG0_PROCFEATR	Processing Element Features CAR	0x0010	R	1139
SRCFG0_SROP	Source Operations CAR	0x0018	R	1140
SRCFG0_DSTOP	Destination Operations CAR	0x001C	R	1141
SRCFG0_MAILBOX	Mailbox CSR	0x0040	R	1142
SRCFG0_WRDBELL	Write Port/Doorbell CSR	0x0044	R	1143
SRCFG0_PROCLAYCTL	Processing Element Logical Layer Control CSR	0x004C	R	1143
SRCFG0_LCFGSPLBASE	Local Configuration Space Low Base Address Register CSR	0x005C	R/W	1143
SRCFG0_BASEDEVID	Base Device ID CSR	0x0060	R/W	1144
SRCFG0_HOBASEDEVID	Host Base Device ID Lock CSR	0x0068	R/W	1144
SRCFG0_CMPTAG	Component Tag CSR	0x006C	R/W	1144
LP-Serial Port Maintenance Block				
SRCFG0_LPSERPOTMBLKHD0	LP-Serial Port Maintenance Block Header 0	0x0100	R	1145
SRCFG0_LPSERPOTLKTOCTL	LP-Serial Port Link Time-out Control CSR	0x0120	R/W	1145
SRCFG0_LPSERPOTRTO	LP-Serial Port Response Time-out Control CSR	0x0124	R/W	1145
SRCFG0_LPSERPOTGENCTL	LP-Serial Port General Control CSR	0x013C	R/W	1146
SRCFG0_LPSERPOTLACKIDSTAT	LP-Serial Port Local AckID Status CSR	0x0148	R/W	1146
SRCFG0_LPSERPOTERRSTAT	LP-Serial Port Error and Status CSR	0x0158	See register	1146
SRCFG0_LPSERPOTCTL	LP-Serial Port Control CSR	0x015C	See register	1147
Extended Error Management (EME) Block				
SRCFG0_EMEBLKHDR0	Extended Error Management (EME) Block Header 0	0x0200	R	1149
SRCFG0_EMELOGTRLAERRDTC	EME Logical/Transport Layer Error Detect CSR	0x0208	R/Clear	1149
SRCFG0_EMELOGTRLAERREN	EME Logical/Transport Layer Error Enable CSR	0x020C	R/W	1150
SRCFG0_EMELOGTRLAADRCAPH	EME Logical/Transport Layer High Address Capture CSR	0x0210	R	1151
SRCFG0_EMELOGTRLAADRCAP	EME Logical/Transport Layer Address Capture CSR	0x0214	R	1151
SRCFG0_EMELOGTRLADEVIDCAP	EME Logical/Transport Layer Device ID Capture CSR	0x0218	R	1151
SRCFG0_EMELOGTRLACTLCAP	EME Logical/Transport Layer Control Capture CSR	0x021C	R	1152
SRCFG0_EMEPOTERRDTC	EME Port Error Detect CSR	0x0240	R/Clear	1152
SRCFG0_EMEPOTERRRATEEN	EME Port Error Rate Enable CSR	0x0244	R/W	1153

Table 32-9. SRIO CARs/CSRs (Continued)

Mnemonic	Description	Offset	Access	Page
SRCFG0_EMEPOTATTERRCAP	EME Port Attributes Error Capture CSR	0x0248	R	1154
SRCFG0_EMEPOTPKCTLERRCAP	EME Port Packet/Control Symbol Error Capture CSR	0x024C	R	1154
SRCFG0_EMEPOTERRCAP1	EME Port Error Capture CSR 1	0x0250	R	1154
SRCFG0_EMEPOTERRCAP2	EME Port Error Capture CSR 2	0x0254	R	1154
SRCFG0_EMEPOTERRCAP3	EME Port Error Capture CSR 3	0x0258	R	1155
SRCFG0_EMEPOTERRRATE	EME Port Error Rate CSR	0x0268	See register	1155
SRCFG0_EMEPOTERRRATETD	EME Port Error Rate Threshold CSR	0x026C	R/W	1156
RLL Extended CAR/CSR Descriptions				
SRCFG0_ECARCTL	RLL ECAR Control CSR	0x10000	R/W	1156
SRCFG0_ECARR2BMBPTR0H	RLL ECAR R2B Mailbox Pointer 0 (Hi_Add) CSR	0x10010	R/W	1157
SRCFG0_ECARR2BMBPTR0L	RLL ECAR R2B Mailbox Pointer 0 (Lo_Add) CSR	0x10014	R/W	1157
SRCFG0_ECARR2BMBPTR1H	RLL ECAR R2B Mailbox Pointer 1 (Hi_Add) CSR	0x10018	R/W	1157
SRCFG0_ECARR2BMBPTR1L	RLL ECAR R2B Mailbox Pointer 1 (Lo_Add) CSR	0x1001C	R/W	1158
SRCFG0_ECARR2BMBPTR2H	RLL ECAR R2B Mailbox Pointer 2 (Hi_Add) CSR	0x10020	R/W	1158
SRCFG0_ECARR2BMBPTR2L	RLL ECAR R2B Mailbox Pointer 2 (Lo_Add) CSR	0x10024	R/W	1158
SRCFG0_ECARR2BMBPTR3H	RLL ECAR R2B Mailbox Pointer 3 (Hi_Add) CSR	0x10028	R/W	1158
SRCFG0_ECARR2BMBPTR3L	RLL ECAR R2B Mailbox Pointer 3 (Lo_Add) CSR	0x1002C	R/W	1159
SRCFG0_ECARDBLINFO	RLL ECAR R2B Doorbell Info CSR	0x10030	R	1159
SRCFG0_ECARDBLSRC	RLL ECAR R2B Doorbell SRC CSR	0x10034	R	1159
SRCFG0_ECARB2RADDWINBAS0	RLL ECAR B2R Add Window Base 0	0x10038	R/W	1159
SRCFG0_ECARB2RADDWINBAS1	RLL ECAR B2R Add Window Base 1	0x1003C	R/W	1160
SRCFG0_ECARB2RADDWINLIM0	RLL ECAR B2R Add Window Limit 0	0x10040	R/W	1160
SRCFG0_ECARB2RADDWINLIM1	RLL ECAR B2R Add Window Limit 1	0x10044	R/W	1160
SRCFG0_ECARR2BADDWINBAS0	RLL ECAR R2B Add Window Base 0	0x10048	R/W	1160
SRCFG0_ECARR2BADDWINBAS1	RLL ECAR R2B Add Window Base 1	0x1004C	R/W	1161
SRCFG0_ECARR2BADDWINLIM0	RLL ECAR R2B Add Window Limit 0	0x10050	R/W	1161
SRCFG0_ECARR2BADDWINLIM1	RLL ECAR R2B Add Window Limit 1	0x10054	R/W	1161
SRCFG0_ECARBILEVTINTSTAT	RLL ECAR BIL Event/Interrupt Status	0x10058	See register	1161
SRCFG0_ECARBILINTMASK	RLL ECAR BIL Interrupt Mask	0x1005C	R/W	1164
SRCFG0_ECARBILMBXLRSTAT	RLL ECAR BIL MBX/Letter Status	0x10060	See register	1165
SRCFG0_ECARR2BPOTWRPRMCAP	RLL ECAR R2B Port Write Parameter Capture	0x10068	R	1168
SRCFG0_ECARR2BPOTWRSRCCAP	RLL ECAR R2B Port Write Source Capture	0x1006C	R	1168
SRCFG0_ECARB2RCTLSYM	RLL ECAR B2R Control Symbol	0x10070	R/W	1168
Outbound Base Registers				
SRCFG0_SOMOMR1LAL	SOM Address Low for Memory Region 1	0x12000	R/W	1169
SRCFG0_SOMOMR1LAH	SOM Address High for Memory Region 1	0x12008	R/W	1169
SRCFG0_SOMOMR2LAL	SOM Address Low for Memory Region 2	0x12010	R/W	1169
SRCFG0_SOMOMR2LAH	SOM Address High for Memory Region 2	0x12018	R/W	1169
SRCFG0_SOMOMR3LAL	SOM Address Low for Memory Region 3	0x12020	R/W	1170
SRCFG0_SOMOMR3LAH	SOM Address High for Memory Region 3	0x12028	R/W	1170
SRCFG0_SOMMSGLAL	SOM Address Low for Message Region	0x12030	R/W	1170
SRCFG0_SOMMSGLAH	SOM Address High for Message Region	0x12038	R/W	1170

Table 32-9. SRIO CARs/CSRs (Continued)

Mnemonic	Description	Offset	Access	Page
SRCFG0_SOMCFGLAL	SOM Address Low for Configuration Region	0x12040	R/W	1171
SRCFG0_SOMCFGLAH	SOM Address High for Configuration Region	0x12048	R/W	1171
SRCFG0_SOMMNTLAL	SOM Address Low for Maintenance Region	0x12050	R/W	1171
SRCFG0_SOMMNTLAH	SOM Address High for Maintenance Region	0x12058	R/W	1171
Outbound Destination Location Registers				
SRCFG0_DIDLOMR1	Outbound Destination Location for Memory Region 1	0x12060	R/W	1172
SRCFG0_DIDLOMR2	Outbound Destination Location for Memory Region 2	0x12068	R/W	1172
SRCFG0_DIDLOMR3	Outbound Destination Location for Memory Region 3	0x12070	R/W	1172
SRCFG0_DIDLMSG	Outbound Destination Location for Message Region	0x12078	R/W	1173
SRCFG0_DIDLMNT	Outbound Destination Location for Maintenance Region	0x12088	R/W	1173
Inbound Base and Mask Registers				
SRCFG0_BAR0	Inbound BAR Register 0	0x12090	R/W	1173
SRCFG0_BAR0M	Inbound Mask and Enable Register 0	0x12098	R/W	1174
SRCFG0_SIM01S	Inbound Size Register 01	0x120A0	R/W	1174
SRCFG0_SIM0LAL	Inbound Lower Base Address Register 0	0x120A8	R/W	1174
SRCFG0_SIM0LAH	Inbound Upper Base Address Register 0	0x120B0	R/W	1175
SRCFG0_SIM1LAL	Inbound Lower Base Address Register 1	0x120B8	R/W	1175
SRCFG0_SIM1LAH	Inbound Upper Base Address Register 1	0x120C0	R/W	1175
SRCFG0_BAR1	Inbound BAR Register 1	0x120C8	R/W	1175
SRCFG0_BAR1M	Inbound Mask and Enable Register 1	0x120D0	R/W	1176
SRCFG0_SIM23S	Inbound Size Register 23	0x120D8	R/W	1176
SRCFG0_SIM2LAL	Inbound Lower Base Address Register 2	0x120E0	R/W	1176
SRCFG0_SIM2LAH	Inbound Upper Base Address Register 2	0x120E8	R/W	1177
SRCFG0_SIM3LAL	Inbound Lower Base Address Register 3	0x120F0	R/W	1177
SRCFG0_SIM3LAH	Inbound Upper Base Address Register 3	0x120F8	R/W	1177

32.6.3 SRIO CAR/CSR Descriptions

The following sections describe the SRIO CARs and CSRs in detail.

32.6.3.1 Device IDs CAR (SRCFG0_DEVID)

Figure 32-42.Device ID and Vendor Information (SRCFG0_DEVID)

Bits	Field Name	Reset State	Function
0:15	DeviceID	See description	Device Identifier The value of this field is sourced from SRREG0_ID1[SDID].
16:31	DeviceVendorID	See description	Device Vendor Identifier The value of this field is sourced from SRREG0_ID1[SVID].

32.6.3.2 Device Information CAR (SRCFG0_DEVINFO)

This register contains additional information about the device.

Figure 32-43.Device Information CAR (SRCFG0_DEVINFO)

Bit	Field Name	Reset State	Function
0:15			Reserved
16:23	DeviceRev	See description	Device Revision Level The value of this field is sourced from SRREG0_ID2[DREV].
24:31	Major/Minor Rev	See description	Spec Revision Level Compliance (1.2) The value of this field is sourced from SRREG0_ID2[SREV].

32.6.3.3 Assembly IDs CAR (SRCFG0_ASSEMBID)

This register identifies the specific assembly vendor and device.

Figure 32-44.Assembly IDs CAR (SRCFG0_ASSEMBID)

Bit	Field Name	Reset State	Function
0:15	AssyID	See description	Assembly Identifier The value of this field is sourced from SRREG0_ID3[AID].
16:31	AssyVendorID	See description	Assembly Vendor Identifier The value of this field is sourced from SRREG0_ID3[AVID].

32.6.3.4 Assembly Information CAR (SRCFG0_ASEMBINFO)

This register contains additional information about the assembly.

Figure 32-45.Assembly Information CAR (SRCFG0_ASEMBINFO)

Bit	Field Name	Reset State	Function
0:15	AssyRev	See description	Assembly Revision Level The value of this field is sourced from SRREG0_ID4[AREV].
16:31	Extended Features Pointer	0x0100	Pointer to the first entry in the extended features list

32.6.3.5 Processing Element Features CAR (SRCFG0_PROCFEATR)

This register identifies the major functionality provided by the processing element.

Figure 32-46.Processing Element Features CAR (SRCFG0_PROCFEATR)

Bit	Field Name	Reset State	Function
0	Bridge	0b1	PE can bridge to another interface. Examples are PCI, proprietary processor buses, DRAM, etc. This bit is likely applicable to nearly every SRIO device.
1	Memory	0b0	PE has physically addressable local address space and can be accessed as an end point through non-MAINTENANCE (i.e., Nread and Nwrite) transactions. This local address space may be limited to local configuration registers, or could be on-chip SRAM, etc.
2	Processor	0b0	PE physically contains a local processor or similar device that executes code. A device that bridges to an interface that connects to a processor does not count (see bit 0 above).

Figure 32-46.Processing Element Features CAR (SRCFG0_PROCFEATR) (Continued)

3	Switch	0b0	PE can bridge to another external SRIO interface. An internal port to a local end point does not count as a switch port. For example, a device with two SRIO ports and a local end point is a two port switch, not a three port switch, regardless of the internal architecture.
4:7			Reserved
8	Mailbox 0	0b1	Supports Mailbox 0
9	Mailbox 1	0b1	Supports Mailbox 1
10	Mailbox 2	0b1	Supports Mailbox 2
11	Mailbox 3	0b1	Supports Mailbox 3
12	Doorbell	0b1	Supports inbound Doorbells
13:23			Reserved
24	Flow Control Support	0b0	PE - Logical layer supports Flow extensions (Default value in rio_defs.v, must be set by user to desired value)
25:26			Reserved
27	Common transportlarge system support	0b1	0 - PE does not support common transport large systems 1 - PE supports common transport large systems
28	Extended Features	0b1	PE has Extended Features List; the extended features pointer is valid
29:31	Extended Address Support	0b001	001 - PE only supports 34 bit addresses

32.6.3.6 Source Operations CAR (SRCFG0_SROP)

This register defines the set of SRIO logical operations that can be issued by the RLL. It is assumed that all processing elements can generate maintenance configuration operations in order to access these registers.

Figure 32-47.Source Operations CAR (SRCFG0_SROP)

Bit	Field Name	Reset State	Function
0	Read	0b1	PE can support a read operation
1	Instruction read	0b0	PE can support an instruction read operation
2	Read for ownership	0b0	PE can support a read for ownership operation
3	Data cache invalidate	0b0	PE can support a data cache invalidate operation
4	Castout	0b0	PE can support a castout operation
5	Flush	0b0	PE can support a flush operation
6	I/O read Instruction	0b0	PE can support an I/O read operation
7	Cache invalidate	0b0	PE can support an instruction cache invalidate operation
8	TLB invalidate entry	0b0	PE can support a TLB invalidate entry operation
9	TLB invalidate entry sync	0b0	PE can support a TLB invalidate entry sync operation
10–15			Reserved
16	Nread	0b1	PE can issue an Nread Operation
17	Nwrite	0b1	PE can issue an Nwrite Operation
18	Swrite	0b1	PE can issue an Swrite Operation
19	Nwrite_r	0b1	PE can issue an Nwrite_r Operation
20	Message	0b1	PE can issue a Message Operation
21	Doorbell	0b1	PE can issue a Doorbell Operation
22			Reserved
23	Atomic (Test and Swap)	0b1	PE can issue an Atomic Test and Swap operation

Figure 32-47.Source Operations CAR (SRCFG0_SROP) (Continued)

24	Atomic (Increment)	0b1	PE can issue an Atomic Increment operation
25	Atomic (Decrement)	0b1	PE can issue an Atomic Decrement operation
26	Atomic (Set)	0b1	PE can issue an Atomic Set operation
27	Atomic (Clear)	0b1	PE can service an Atomic Clear operation
28			Reserved
29	Port Write	0b1	PE can service a Port Write
30:31			Reserved

32.6.3.7 Destination Operations CAR (SRCFG0_DSTOP)

This register defines the set of SRIO operations that can be serviced by the RLL. It is assumed that all processing elements can service maintenance configuration read and write maintenance operations in order to access these registers.

Figure 32-48.Destination Operations CAR (SRCFG0_DSTOP)

Bit	Field Name	Reset State	Function
0	Read	0b1	PE can support a read operation
1	Instruction read	0b0	PE can support an instruction read operation
2	Read for ownership	0b0	PE can support a read for ownership operation
3	Data cache invalidate	0b0	PE can support a data cache invalidate operation
4	Castout	0b0	PE can support a castout operation
5	Flush	0b0	PE can support a flush operation
6	I/O read Instruction	0b0	PE can support an I/O read operation
7	Cache invalidate	0b0	PE can support an instruction cache invalidate operation
8	TLB invalidate entry	0b0	PE can support a TLB invalidate entry operation
9	TLB invalidate entry sync	0b0	PE can support a TLB invalidate entry sync operation
10:15			Reserved
16	Nread	0b1	PE can service an Nread Operation
17	Nwrite	0b1	PE can service an Nwrite Operation
18	Swrite	0b1	PE can service an Swrite Operation
19	Nwrite_r	0b1	PE can service an Nwrite_r Operation
20	Message	0b1	PE can service a Message Operation
21	Doorbell	0b1	PE can service a Doorbell Operation
22			Reserved
23	Atomic (Test and Swap)	0b1	PE can service an Atomic Test and Swap operation
24	Atomic (Increment)	0b1	PE can service an Atomic Increment operation
25	Atomic (Decrement)	0b1	PE can service an Atomic Decrement operation
26	Atomic (Set)	0b1	PE can service an Atomic Set operation
27	Atomic (Clear)	0b1	PE can service an Atomic Clear operation
28			Reserved
29	Port Write	0b1	PE can service a Port Write
30:31			Reserved

32.6.3.8 Mailbox CSR (SRCFG0_MAILBOX)

The mailbox command and status register is accessed if an external processing element wishes to determine the status of this processing elements's mailbox hardware, if any is present. It is not necessary to examine this register before sending a message since the SRIO protocol will accept, retry, or send an error response message depending upon the status of the addressed mailbox. This register is read-only.

Figure 32-49.Mailbox CSR (SRCFG0_MAILBOX)

Bit	Field Name	Reset State	Function
0	Mailbox 0 Available	0b1	Mailbox 0 is initialized and ready to accept messages. Always set.
1	Mailbox 0 Full	0b0	Mailbox 0 is full.
2	Mailbox 0 Empty	0b1	Mailbox 0 has no outstanding messages.
3	Mailbox 0 Busy	0b0	Mailbox 0 is busy receiving a message operation.(See RLL_IP_messaging overview for implementation details)
4	Mailbox 0 Failed	0b0	Never set
5	Mailbox 0 Error	0b0	Mailbox 0 had an internal fault or error condition and is waiting for assistance.
6:7			Reserved
8	Mailbox 1 Available	0b1	Mailbox 1 is initialized and ready to accept messages. Always set.
9	Mailbox 1 Full	0b0	Mailbox 1 is full.
10	Mailbox 1 Empty	0b1	Mailbox 1 has no outstanding messages.
11	Mailbox 1 Busy	0b0	Mailbox 1 is busy receiving a message operation.
12	Mailbox 1 Failed	0b0	Never set
13	Mailbox 1 Error	0b0	Mailbox 1 had an internal fault or error condition and is waiting for assistance.
14:15			Reserved
16	Mailbox 2 Available	0b1	Mailbox 2 is initialized and ready to accept messages.Always set.
17	Mailbox 2 Full	0b0	Mailbox 2 is full.
18	Mailbox 2 Empty	0b1	Mailbox 2 has no outstanding messages.
19	Mailbox 2 Busy	0b0	Mailbox 2 is busy receiving a message operation.
20	Mailbox 2 Failed	0b0	Never set
21	Mailbox 2 Error	0b0	Mailbox 2 had an internal fault or error condition and is waiting for assistance.
22:23			Reserved
24	Mailbox 3 Available	0b1	Mailbox 3 is initialized and ready to accept messages. Always set.
25	Mailbox 3 Full	0b0	Mailbox 3 is full.(See RLL_IP_messaging overview for implementation details)
26	Mailbox 3 Empty	0b1	Mailbox 3 has no outstanding messages.
27	Mailbox 3 Busy	0b0	Mailbox 3 is busy receiving a message operation.
28	Mailbox 3 Failed	0b0	Never set
29	Mailbox 3 Error	0b0	Mailbox 3 had an internal fault or error condition and is waiting for assistance.(See RLL_IP_messaging overview for implementation details)
30:31			Reserved

32.6.3.9 Write Port and Doorbell CSR (SRCFG0_WRDBELL)

This register contains the status of the R2B doorbell hardware and the R2B Port Write hardware.

Figure 32-50.Write Port/Doorbell CSR (SRCFG0_WRDBELL)

Bit	Field Name	Reset State	Function
0	Doorbell Available	0b1	Always set.
1	Doorbell Full	0b0	Doorbell is full. All incoming message packets return retry responses.
2	Doorbell Empty	0b1	Doorbell has no outstanding messages.
3	Doorbell Busy	0b0	Doorbell is busy receiving a message operation. New message operations return retry responses.
4	Doorbell Failed	0b0	Never set.
5	Doorbell Error	0b0	Doorbell had an internal fault or error condition and is waiting for assistance.
6:23			Reserved
24	WR_Port Available	0b1	Always set
25	WR_Port Full	0b0	WR_Port is full. This bit reflects the Port Write occurred bit in the ECAR BIL Interrupt Status CSR Further incoming port writes are allowed and will be executed and the trapped port write info CAS/CSR's will be overwritten.
26	WR_Port Empty	0b1	WR_Port is empty. This bit reflects the inverted value of the Port Write occurred bit in the ECAR BIL Interrupt Status CSR
27	WR_Port Busy	0b0	WR_Port is busy receiving a Port-write operation. This bit reflects the Port Write occurred bit in the ECAR BIL Interrupt Status CSR. Further incoming port writes are allowed and will be executed and the trapped port write info CAS/CSR's will be overwritten.
28	WR_Port Failed	0b0	Never set. There are no events which are considered as Failed by the WR_Port hardware.
29	WR_Port Error	0b0	Never set. There are no events which are considered as HW ERROR by the WR_Port hardware.
30:31			Reserved

32.6.3.10 Processing Element Logical Layer Control CSR (SRCFG0_PROCLAYCTL)

The Processing Element Logical Layer Control CSR is used for general command and status information for the logical interface. This register is read-only.

Figure 32-51.PE Logical Layer Control CSR (SRCFG0_PROCLAYCTL)

Bit	Field Name	Reset State	Function
0:28			Reserved
29:31	Extended addressing control	0b001 or 0b100	Indicates the number of address bits generated by the RLL as a source and processed by the RRL as the target of an operation. Controlled by parameter in Rll_Defs. 0b100 - PE supports 66 bit addresses (when RIO_ADDR_SIZE = 66) 0b001 - PE supports 34 bit addresses (when RIO_ADDR_SIZE = 34)

32.6.3.11 LCSLow Base Addr Register CSR (SRCFG0_LCFGSPLBASE)

The contents of this register are utilized to allow the mapping of the RLL CAR/CSR space into a window of inbound SRIO memory space. This window occupies 2M of Mod64 addresses (16M of Byte addresses) and may be located at any address which is an even multiple of the window size. Inbound SRIO Nwrite, Swrite, Nwrite_R and Nread_R transactions, with an address that match the valid LCSBA comparison bits will be vectored to the RLL internal CAR/CSR controller and not passed out the BIL R2B bus to the user.

Figure 32-52.LCS Low Base Address Register CSR (SRCFG0_LCFGSPLBASE)

Bit	Field Name	Reset State	Function
0			Reserved
1–31	LCSBA	0x1E000000	Maintence (CAR/CSR) space relocation Base Address value. Bit 1 maps to Addr 33 of inbound SRIO address ... Bit 31 maps to Addr 3 of inbound SRIO address Note: that Bit 1 through Bit 10 (Addr 33 – Addr 24) are utilized by the inbound SRIO address comparison logic. Bits 11–31 are ignored. (Default value in rio_defs.v, currently configured to set LCSBA for an inbound R2B address of: rmp at 0x0_f000_0000)

32.6.3.12 Base Device ID CSR (SRCFG0_BASEDEVID)

The base device ID CSR contains the base device ID values for the processing element.

Figure 32-53.Base Device ID CSR (SRCFG0_BASEDEVID)

Bit	Field Name	Reset State	Function
0:7			Reserved
8:15	Base_deviceID	0xFF	This is the base ID of the device in a small common transport system. (Default value specified in rio_defs.v)
16:31	Large_Base_deviceID	0xFFFF	This is the base ID of the device in a large common transport system (only valid for end point device and if bit 27 of the Processing Element Features CAR is set). (Default value specified in rio_defs.v)

32.6.3.13 Host Base Device ID Lock CSR (SRCFG0_HOBASEDEVID)

The host base device ID lock CSR contains the base device ID value for the processing element in the system that is responsible for initializing this processing element. The Host_base_deviceID field is a write-once/reset-able field which provides a lock function. Once the Host_base_deviceID field is written, all subsequent writes to the field are ignored, except in the case that the value written matches the value contained in the field. In this case, the register is re-initialized to 0xFFFF. After writing the Host_base_deviceID field a processing element must then read the Host Base Device ID Lock CSR to verify that it owns the lock before attempting to initialize this processing element.

Figure 32-54.Host Base Device ID Lock CSR (SRCFG0_HOBASEDEVID)

Bit	Field Name	Reset State	Function
0:15			Reserved
16:31	Host_base_deviceID	0xFFFF	This is the base device ID for the PE that is initializing this PE.

32.6.3.14 Component Tag CSR (SRCFG0_CMPTAG)

The component tag CSR contains a component tag value for the processing element and can be assigned by software when the device is initialized. It is especially useful for labeling and identifying devices that are not end points and do not have device ID registers.

Figure 32-55.Component Tag CSR (SRCFG0_CMPTAG)

Bit	Field Name	Reset State	Function
0:31	Component_tag	0x00000000	This is a component tag for the PE.

32.6.4 LP-Serial Port Maintenance Block

The RLL has register definitions for one SRIO LP-Serial port embedded in it. The port is numbered 0. This block is defined for a LP-Serial Generic endpoint device with one exception. The LP-Serial Port Local AckID Status CSR has been included in the block, with limited support, to facilitate hot-swap functionality.

32.6.4.1 LP-Serial Port Maintenance Block Header 0 (SRCFG0_LPSERPOTMBLKHD0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the port maintenance block header.

Figure 32-56.LP-Serial Port Maintenance Block Header (SRCFG0_LPSERPOTMBLKHD0)

Bit	Field Name	Reset State	Function
0:15	EF_PTR	0x0200	Pointer to EME block data structure
16:31	EF_ID	0x0001[1]	Hard wired Extended Features ID
Note 1: This is the LP-Serial Generic Endpoint Block ID as redefined in Rev 1.0 of the SRIO Errata.			

32.6.4.2 LP-Serial Port Link Timeout Control CSR (SRCFG0_LPSERPOTLKTOCTL)

The port link timeout control register contains the timeout timer value for all ports on a device. This timeout is for link events such as sending a packet to receiving the corresponding acknowledge, and sending a link-request to receiving the corresponding link-response. The reset value is the maximum timeout interval, and represents between 3 and 5 seconds. The timer granularity is TBD (between 180ns and 300ns). This is considered to be a PHY-Transport level timer, and, as such, is routed to the R2_IP_SER3 IP block.

Figure 32-57.LP-Serial Port Link Timeout CTL CSR (SRCFG0_LPSERPOTLKTOCTL)

Bit	Field Name	Reset State	Function
0:23	PhyTrn-Timeout value	0x00000C	Reset value controlled by RLL_LINK_TMO_VAL in rio_defs.v (RLL_LINK_TMO_VAL parameter). The Reset state shown here is the default value in the distribution rio_defs.v and may be reassigned to any value by the user – in their local rio_defs.v file.
24:31			Reserved

32.6.4.3 LP-Serial Port Response Time-out Control CSR (SRCFG0_LPSERPOTRTO)

The port response timeout control register contains the timeout timer count for all ports on a device. This timeout is for sending a request packet to receiving the corresponding response packet. The reset value is the maximum timeout interval, and represents between 3 and 5 seconds. The timer granularity is TBD (between 180ns and 300ns). This is considered to be a logical level timer, and, as such, is routed to the BIL for use by the user logic in determining logical transaction timeout conditions.

Figure 32-58.LP-Serial Port Response Timeout CTL CSR

Bit	Field Name	Reset State	Function
0:23	Logical-Timeout value	0x010000	Reset value controlled by RLL_LOGICAL_TMO_VAL in rio_defs.v. (RLL_LOGICAL_TMO_VAL parameter) The Reset state shown here is the default value in the distribution rio_defs.v and may be reassigned to any value by the user – in their local rio_defs.v file.
24:31			Reserved

32.6.4.4 LP-Serial Port General Control CSR (SRCFG0_LPSERPOTGENCTL)

The port general control register contains control register bits applicable to all ports on a processing element.

Figure 32-59.LP-Serial Port General Control CSR (SRCFG0_LPSERPOTGENCTL)

Bit	Field Name	Reset State	Function
0	Host	0b0	A Host device is a device that is responsible for system exploration, initialization, and maintenance. Agent or slave devices are initialized by Host devices. 0 - agent or slave device 1 - host device
1	Master Enable	0b1	The Master Enable bit controls whether or not a device is allowed to issue requests into the system. If the Master Enable is not set, the device may only respond to requests. 0 - processing element cannot issue requests 1 - processing element can issue requests The Reset state shown here is the default value in the distribution rio_defs.v and may be reassigned to any value by the user – in their local rio_defs.v file
2	Discovered	0b0	This device has been located by the processing element responsible for system configuration. 0 - The device has not been previously discovered 1 - The device has been discovered by another processing element
3:31			Reserved

32.6.4.5 LP-Serial Port Local AckID Status CSR (SRCFG0_LPSERPOTLACKIDSTAT)

The port link local ackID status registers are accessible both by a local processor and an external device. A read to this register returns the local ackID status for both the out and input ports of the device. This register has only limited support designated for hot-swap use. Upon reading, the ackID fields will always return a value of zero. Upon writing any 0x0000 to this register, the HW causes R2_IP_SER3 to initialize the Inbound_ackID, Outstanding_ackID, and Outbound_ackID to 0x0, in the R2_IP_SER3 IP Block. For details on how this register would be utilized during the Hot-Swap process, refer to the Mercury R2_IP_SER3 Hardware Specification and Users Guide.

Figure 32-60.LP-Serial Port Local AckID Status CSR (SRCFG0_LPSERPOTLACKIDSTAT)

Bit	Field Name	Reset State	Function
0:2			Reserved
3:7	Inbound ackID	0b00000	Always 0b00000
8:18			Reserved
19:23	Outstanding ackID	0b00000	Always 0b00000
24:26			Reserved
27:31	Outbound ackID	0b00000	Always 0b00000

32.6.4.6 LP-Serial Port Error and Status CSR (SRCFG0_LPSERPOTERRSTAT)

These registers are accessed when a local processor or an external device wishes to examine the port error and status information.

Figure 32-61.LP-Serial Port Error and Status CSR (SRCFG0_LPSERPOTERRSTAT)

Bit	Field Name	Reset State	Function

Figure 32-61.LP-Serial Port Error and Status CSR (SRCFG0_LPSERPOTERRSTAT) (Continued)

0:5			Reserved
6	Output Failed-encountered[1]	0b0	Output Port has Encountered a failed condition, meaning that the port's failed error threshold has been reached in the Port Error Rate Threshold register. (write 1 to clear) R/C
7	Output Degraded-encountered[1]	0b0	Output Port has entered a degraded condition, meaning that the port's degraded error threshold has been reached in the Port Error Rate Threshold register. (write 1 to clear) R/C
8:10			Reserved
11	Output Retried-encountered	0b0	Output port has encountered a retried condition. This bit is set when bit 13 is set (write 1 to clear). R/C
12	Output Blocked	0b0	Output port has received a packet-retry control symbol and can't make forward progress. This bit is set when bit 13 is set and cleared on receiving a packet-accepted or packet-not-accepted control symbol. R
13	Output Retry-stopped	0b0	Output port has received a packet-retry control symbol and is in the "output retry-stopped" state. R
14	Output Error-encountered	0b0	Output port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 15 is set (write 1 to clear). R/C
15	Output Error-stopped	0b0	Output port is in the "output error-stopped" state. R
16:20			Reserved
21	Input Retry-stopped	0b0	Input port is in the "input retry-stopped" state. R
22	Input Error-encountered	0b0	Input port has encountered (and possibly recovered from) a transmission error. This bit is set when bit 23 is set (write 1 to clear). R/C
23	Input Error-Stopped	0b0	Input port is in the "input error-stopped" state. R
24:26			Reserved
27	Port Write Pending	0b0	Port has encountered a condition which required it to initiate a Maint-Port-Write operation. This bit is only Valid if the device is capable of issuing a maintenance port-write transaction. Once set it will remain set until written with a 1 to clear. R/C
28			Reserved
29	Port Error	0b0	Input or output port has encountered an error from which the hardware was unable to recover (write 1 to clear). R/C
30	Port OK	0b0	Input and output ports are initialized and can communicate with the adjacent device. This bit and bit 31 are mutually exclusive. R
31	Port Un-initialized	0b1	Input and output ports are not initialized and is in training mode. This bit and bit 30 are mutually exclusive. R
Note: 1: Bit added per Error Management Extension Specification – only supported in Phase 1B IP.			

32.6.4.7 LP-Serial Port Control CSR (SRCFG0_LPSERPOTCTL)

The port control register contains control register bits for the RLL SRIO port.

Figure 32-62.LP- Serial Port Control CSR (SRCFG0_LPSERPOTCTL)

Bit	Field Name	Reset State	Function
0:1	Output Port Width	0b01	Width of the port: (Default specified in rio_defs.v) 0b00 - Single lane 0b01 - Four Lane 0b10 - 0b11 reserved R
2:4	Initialized Port Width	Implementation dependent	Width of port after initialization: 0b000 - Single Lane, lane 0 0b001 - Single Lane, lane 2 0b010 - Four Lane, 0b011 - 0b111 reserved R
5:7	Port Width override	0b000	Soft port configuration to override the hardware size: (Default specified in rio_defs.v) 0b000 - No override 0b001 - reserved 0b010 - Single Lane, lane 0 0b011 - Single Lane, lane 2 0b100 - 0b111 reserved R/C
8	Port Disable	0b1	Port Disable: 0 - Port drivers and receivers are enabled 1 - Port drivers and receivers are disabled R/W
9	Output Port Enable	0b1	Output port transmit enable: 0 - port is stopped and not enabled to issue any packets except to respond to I/O logical Maintenance packets 1 - port is enabled to issue any packets R/W
10	Input Port Enable	0b1	Input port receive enable: 0 - port is stopped and only enabled to respond I/O logical Maintenance requests. Other requests return packet-not-accepted control symbols to force an error condition to be signaled by the sending device 1 - port is enabled to respond to any packet R/W
11	Error Checking Disable	0b0	This bit disables all SRIO transmission error checking 0 - Error checking and recovery is enabled 1 - Error checking and recovery is disabled Device behavior when error checking and recovery is disabled and an error condition occurs is undefined R/W
12:27			Reserved
28	Stop on Port Failed-encountered enable[1]	0b0	When set causes the port to stop attempting to send packets to the connected device when the Output Failed-encountered bit is set. Packets are discarded if the Drop Packet bit is set. When cleared the port continues to attempt to transmit packets to the connected device if the Output Failed-encountered bit is set. R/W
29	Drop Packet enable[1]	0b0	R/W
30	Port lockout[1]	0b0	R/W
31	Port Type	0b1	Serial port R
Note: 1. Bit added per Error Management Extension Specification.			

32.6.5 Extended Error Management (EME) Block

The following are the RLL definitions for the Extended Error Management register block. These are only supported with read only, reset default value access in phase 1A of the RLL IP. Full event detection/status/control with these registers is supported in phase 1B of the RLL IP.

32.6.5.1 EME Block Header 0 (SRCFG0_EMEBLKHDR0)

The port maintenance block header 0 register contains the EF_PTR to the next EF_BLK and the EF_ID that identifies this as the port maintenance block header.

Figure 32-63.EME Block Header 0 CSR (SRCFG0_EMEBLKHDR0)

Bit	Field Name	Reset State	Function
0:15	EF_PTR	0x0000	No next block in the data structure.
16:31	EF_ID	0x0007	Hard wired Extended Features ID.

32.6.5.2 EME Logical/Transport Error Detect CSR (SRCFG0_EMELOGTRLAERRDTC)

This register contains the status bits for errors detected in the Transport and Logical layers. This register is defined by the SRIO Error Management Extensions Specification. Additional Logical layer error status bits, specific to the RLL are contained in the RLL ECAR registers.

Figure 32-64.EME Log/Trnsp Err Detect CSR (SRCFG0_EMELOGTRLAERRDTC)

Bit	Field Name	Reset State	Function
0	IO error response	0b0	Received a response of ERROR for an I/O Logical Layer B2R Request. Generated from user/BIL as processing of inbound response occurs. Caused by the User asserting rll_bil_io_err_rsp_evnt on the BIL interface.
1	Message error response	0b0	Received a response of ERROR for an MSG Logical Layer B2R Request. Generated from user/BIL as processing of inbound response occurs. Caused by the User asserting rll_bil_msg_err_rsp_evnt on the BIL interface.
2			Reserved
3	Message Format error	0b0	Received MESSAGE packet payload with an invalid size or segment (MSG Logical). Reported only if packet discarded (see Message handling).
4	Illegal transaction decode	0b0	Received illegal fields in the request/response packet for a supported transaction (IO/MSG/GSM logical)[1]. Set internally by RLL, as it parses a transaction. Packet responded to with ERROR.
5	Illegal transaction target error	0b0	Received a packet that contained a destination ID that is not defined for this end point. If Promiscuous mode this bit is NEVER set, all R2B packets will be accepted.
6	Message Request Timeout	0b0	A required message request has not been received within the specified timeout interval (MSG logical).
7	Packet Response Timeout	0b0	A required response has not been received within the specified time out interval (IO/MSG/GSM logical). Generated from user/BIL as processing of inbound response occurs. Caused by the User asserting rll_bil_pkt_rsp_tmout_evnt on the BIL interface.
8	Unsolicited Response	0b0	An unsolicited/unexpected Response packet was received (IO/MSG/GSM logical). Generated from user/BIL as processing of inbound response occurs. Caused by the User asserting rll_bil_unslctd_rsp_evnt on the BIL interface.
9	Unsupported Transaction (R2B)	0b0	A transaction is received from SRIO that is not supported in the Destination Operations CAR (IO/MSG/GSM logical). Set internally by RLL, as it parses a transaction. Packet responded to with ERROR.
10:23			Reserved

Figure 32-64.EME Log/Trnsp Err Detect CSR (SRCFG0_EMELOGTRLAERRDTC) (Continued)

24	Unsupported Transaction (B2R)	0b0	A transaction is received from the User through the B2R BIL that is not supported in the Source Operations CAR (IO/MSG/GSM logical). Set internally by RLL, as it parses a transaction. User/BIL request, responded to with ERROR.
25	Unsupported Address (R2B)	0b0	A transaction is received from SRIO that contained an address outside of the R2B Address Window Base/Limit register values. Set internally by RLL, as it parses a transaction. Packet responded to with ERROR.
26	Unsupported Address (B2R)	0b0	A transaction is received from the User through the B2R BIL that contained an address outside of the B2R Address Window Base/Limit register values. Set internally by RLL, as it parses a transaction. User/BIL request, responded to with ERROR.
24:31			Reserved
Note 1: The definition of "illegal fields" covers: - Invalid TTYPE, for a given TYPE on an R2B request. - Data in Request Payload, on Transaction that should not have data. - Data in Request Payload, greater than that specified by inbound SIZE/STS field in packet.			

32.6.5.3 EME Logical/Transport Error Enable CSR (SRCFG0_EMELOGTRLAERREN)

This register contains the bits that control if an error condition locks the Logical/Transport Layer Error Detect and Capture registers and is reported to the system host.

Figure 32-65.EME Log/Trnsp Err Enable CSR (SRCFG0_EMELOGTRLAERREN)

Bit	Field Name	Reset State	Function
0	IO error response enable	0b0	Enable capture and reporting of a Received a response of ERROR for an I/O Logical Layer Request.
1	Message error response enable	0b0	Enable capture and reporting of a Received a response of ERROR for an MSG Logical Layer Request.
2			Reserved
3	Message Format error enable	0b0	Enable capture and reporting of a Received MESSAGE packet payload with an invalid size or segment (MSG Logical).
4	Illegal transaction decode enable	0b0	Enable capture and reporting of a Received illegal fields in the request/response packet for a supported transaction (IO/MSG/GSM logical).
5	Illegal transaction target error enable	0b0	Enable capture and reporting of a Received a packet that contained a destination ID that is not defined for this end point. End points with multiple ports and a built-in switch function may not report this as an error (Transport).
6	Message Request Timeout enable	0b0	Enable capture and reporting of a required message request has not been received within the specified timeout interval (MSG logical).
7	Packet Response Timeout enable	0b0	Enable capture and reporting of a required response has not been received within the specified time out interval (IO/MSG/GSM logical).
8	Unsolicited Response enable	0b0	Enable capture and reporting of An unsolicited/unexpected Response packet was received (IO/MSG/GSM logical; only Maintenance response for switches).
9	Unsupported Transaction enable (R2B)	0b0	Enable capture and reporting of a transaction is received that is not supported in the Destination Operations CAR (IO/MSG/GSM logical.
10:23			Reserved
24	Unsupported Transaction enable (B2R)	0b0	Enable capture and reporting of a transaction is received from the User through the B2R BIL that is not supported in the Source Operations CAR (IO/MSG/GSM logical).
25	Unsupported Address enable (R2B)	0b0	Enable capture and reporting of a transaction is received from SRIO that contained an address outside of the R2B Address Window Base/Limit register values.
26	Unsupported Address enable (B2R)	0b0	Enable capture and reporting of a transaction is received from the User through the B2R BIL that contained an address outside of the B2R Address Window Base/Limit register values.
27:31			Reserved

32.6.5.4 EME Log/Trnsp High Address Capture CSR (SRCFG0_EMELOGTRLAADRCAPH)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set. This register is only required for end point devices that support 66- or 50-bit addresses.

Figure 32-66.EME Log/Trnsp High AddrCap CSR (SRCFG0_EMELOGTRLAADRCAPH)

Bit	Field Name	Reset State	Function
0:31	address [0:31]	0x00000000	Most significant 32 bits of the address associated with the error (for requests, for responses if available).

32.6.5.5 EME Logical/Transport Address Capture CSR (SRCFG0_EMELOGTRLAADRCAP)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set.

Figure 32-67.EME Log/Trnsp AddrCap CSR (SRCFG0_EMELOGTRLAADRCAP)

Bit	Field Name	Reset State	Function
0:28	address [32:60]	0x0000000	Least significant 29 bits of the address associated with the error (for requests, for responses if available).
29			Reserved
30:31	xambs	0b00	Extended address bits of the address associated with the error (for requests, for responses if available).

32.6.5.6 EME Logical/Transport Device ID Capture CSR (SRCFG0_EMELOGTRLADEVIDCAP)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set.

Figure 32-68.EME Log/Trnsp DeviceID Cap CSR (SRCFG0_EMELOGTRLADEVIDCAP)

Bit	Field Name	Reset State	Function
0:7	MSB destinationID	0x00	Most significant byte of the destinationID associated with the error (large transport systems only).
8:15	destinationID	0x00	The destinationID associated with the error.
16:23	MSB sourceID	0x00	Most significant byte of the sourceID associated with the error (large transport systems only).
24:31	sourceID	0x00	The sourceID associated with the error.

32.6.5.7 EME Logical/Transport Control Capture CSR (SRCFG0_EMELOGTRLACTLCAP)

This register contains error information. It is locked when a Logical/Transport error is detected and the corresponding enable bit is set.

Figure 32-69.EME Log/Trnsp Control Cap CSR (SRCFG0_EMELOGTRLACTLCAP)

Bit	Field Name	Reset State	Function
0:3	ftype	0x0	Format type associated with the error.
4:7	ttype	0x0	Transaction type associated with the error.
8:15	msg info	0x00	Letter, mbox, and msgseg for the last Message request received for the mailbox that had an error (Message errors only).
16:31			Reserved

32.6.5.8 EME Port Error Detect CSR (SRCFG0_EMEPOTERRDTC)

The Port Error Detect Register indicates transmission errors that are detected by the hardware.

Figure 32-70.EME Port Error Detect CSR (SRCFG0_EMEPOTERRDTC)

Bit	Field Name	Reset State	Function
0:8			Reserved
9	Received corrupt control symbol	0b0	Received a control symbol with a bad CRC value (serial).
10	Received acknowledge control symbol with unexpected ackID	0b0	Received an acknowledge control symbol with an unexpected ackID (packet-accepted or packet_retry).
11	Received packet-not-accepted control symbol	0b0	Received packet-not-accepted acknowledge control symbol.
12	Received packet with unexpected ackID	0b0	Received packet with unexpected ackID value - out-of-sequence ackID.
13	Received packet with bad CRC	0b0	Received packet with a bad CRC value.
14	Received packet exceeds 276 Bytes	0b0	Received packet which exceeds the maximum allowed size.
15:25			Reserved
26	Non-outstanding ackID	0b0	Link_response received with an ackID that is not outstanding.
27	Protocol error	0b0	An unexpected packet or control symbol was received.
28			Reserved
29	Delineation error	0b0	Received unaligned /SC/ or /PD/ or undefined code-group (serial).
30	Unsolicited acknowledge control symbol	0b0	An unexpected acknowledge control symbol was received.
31	Link timeout	0b0	An acknowledge or link-response control symbol is not received within the specified timeout interval.

32.6.5.9 EME Port Error Enable CSR (SRCFG0_EMEPOTERRRATEEN)

This register contains the bits that control when an error condition is allowed to increment the error rate counter in the Port Error Rate Threshold Register and lock the Port Error Capture registers.

Figure 32-71.EME Port Error Enable CSR (SRCFG0_EMEPOTERRRATEEN)

Bit	Field Name	Reset State	Function
0:8			Reserved
9	Received corrupt control symbolerror enable	0b0	Enable error rate counting of Received a control symbol with a bad CRC value (serial).
10	Received acknowledge control symbol with unexpected ackID error enable	0b0	Enable error rate counting of Received an acknowledge control symbol with an unexpected ackID (packet-accepted or packet_retry).
11	Received packet-not-accepted control symbol error enable	0b0	Enable error rate counting of Received packet-not-accepted acknowledge control symbol.
12	Received packet with unexpected ackID error enable	0b0	Enable error rate counting of Received packet with unexpected ackID value – out-of-sequence ackID.
13	Received packet with bad CRC error enable	0b0	Enable error rate counting of Received packet with a bad CRC value.
14	Received packet exceeds 276 Bytes error enable	0b0	Enable error rate counting of Received packet which exceeds the maximum allowed size.
15-25			Reserved
26	Non-outstanding ackID error enable	0b0	Enable error rate counting of Link_response received with an ackID that is not outstanding.
27	Protocol error enable	0b0	Enable error rate counting of an unexpected packet or control symbol was received.
28			Reserved
29	Delineation error enable	0b0	Enable error rate counting of Received unaligned /SC/ or /PD/ or undefined code-group (serial).
30	Unsolicited acknowledge control symbol	0b0	Enable error rate counting of an unexpected acknowledge control symbol was received.
31	Link timeout error enable	0b0	Enable error rate counting of an acknowledge or link-response control symbol is not received within the specified timeout interval.

32.6.5.10 EME Port Attributes Error Capture CSR (SRCFG0_EMEPOTATTERRCAP)

The error capture attribute register indicates the type of information contained in the Port error capture registers. In the case of multiple detected errors during the same clock cycle one of the errors must be reflected in the Error type field.

Figure 32-72.EME Port Attribute Error Capture CSR (SRCFG0_EMEPOTATTERRCAP)

Bit	Field Name	Reset State	Function
0:1	Info type	0b00	Type of information logged. 00 - packet 01 - control symbol (only error capture register 0 is valid) 10 - implementation specific (capture register contents are implementation specific) 11 - undefined (S-bit error), capture as if a packet (parallel physical layer only)
2			Reserved
3:7	Error type	0x00	Encoded value of captured error bit in the Port Error Detect Register.
8:30			Reserved
31	Capture valid info	0b0	This bit is set by hardware to indicate that the Packet/control symbol capture registers contain valid information. For control symbols, only capture register 0 will contain meaningful information.

32.6.5.11 ME Port Packet/CtlSym Error Capture CSR (SRCFG0_EMEPOTPKCTLERRCAP)

Captured control symbol information includes the true and complement of the control symbol. This is exactly what arrives on the SRIO interface with bits 0:7 of the capture register containing the least significant byte of the 32-bit quantity. This register contains the first four bytes of captured packet symbol information.

Figure 32-73.EME Port Packet/CtlSym Error Capture CSR (SRCFG0_EMEPOTPKCTLERRCAP)

Bit	Field Name	Reset State	Function
0:31	Capture 0	All 0s	Control Character and Control Symbol (serial) or bytes 0:3 of Packet Header.

32.6.5.12 EME Port Error Capture CSR 1 (SRCFG0_EMEPOTERRCAP1)

Error Capture Register 1 contains bytes 4:7 of the packet header.

Figure 32-74.EME Port Error Capture CSR 1 (SRCFG0_EMEPOTERRCAP1)

Bit	Field Name	Reset State	Function
0:31	Capture 1	All 0s	Bytes 4:7 of the packet header.

32.6.5.13 EME Port Error Capture CSR 2 (SRCFG0_EMEPOTERRCAP2)

Error Capture Register 2 contains bytes 8:11 of the packet header.

Figure 32-75.EME Port Error Capture CSR 2 (SRCFG0_EMEPOTERRCAP2)

Bit	Field Name	Reset State	Function
0:31	Capture 2	All 0s	Bytes 8:11 of the packet header.

32.6.5.14 EME Port Error Capture CSR 3 (SRCFG0_EMEPOTERRCAP3)

Error Capture Register 3 contains bytes 12:15 of the packet header.

Figure 32-76.EME Port Error Capture CSR 3 (SRCFG0_EMEPOTERRCAP3)

Bit	Field Name	Reset State	Function
0:31	Capture 3	All 0s	Bytes 12:15 of the packet header.

32.6.5.15 EME Port Error Rate CSR (SRCFG0_EMEPOTERRRATE)

The Port Error Rate register is a 32-bit register used with the Port Error Rate Threshold register to monitor and control the reporting of transmission errors.

Figure 32-77.EME Port Error Rate CSR (SRCFG0_EMEPOTERRRATE)

Bit	Field Name	Reset State	Function
0:7	Error Rate Bias	0x00	These bits provide the error rate bias value. 0x00 - do not decrement the error rate counter 0x01 - decrement every 1ms (±34%) 0x02 - decrement every 10ms (±34%) 0x04 - decrement every 100ms (±34%) 0x08 - decrement every 1s (±34%) 0x10 - decrement every 10s (v34%) 0x20 - decrement every 100s (±34%) 0x40 - decrement every 1000s (±34%) 0x80 - decrement every 10000s (±34%) Other values are Reserved R/W
8:13			Reserved
14:15	Error Rate Recovery	0b00	These bits limit the incrementing of the error rate counter above the failed threshold trigger. 0b00 - Only count 2 errors above 0b01 - Only count 4 errors above 0b10 - Only count 16 error above 0b11 - Do not limit incrementing the error rate count R/W
16:23	Peak Error Rate	0x00	This field contains the peak value attained by the error rate counter. R/W
24:31	Error Rate Counter	0x00000000	These bits maintain a count of the number of transmission errors that have been detected by the port. This number is decremented by the Error Rate Bias mechanism. Read only

32.6.5.16 EME Port Error Rate Threshold CSR (SRCFG0_EMEPOTERRRATETD)

The Port Error Rate Threshold register is a 32-bit register used to control the reporting of the link status to the system host.

Figure 32-78.EME Port Error Rate Threshold CSR (SRCFG0_EMEPOTERRRATETD)

Bit	Field Name	Reset State	Function
0:7	Error Rate Failed Threshold Trigger	0xFF	These bits provide the threshold value for reporting an error condition due to a possibly broken link. 0x00 - Disable the error rate register 0x01 - Set the error reporting threshold to 1 0x02 - Set the error reporting threshold to 2 ... 0xFF - Set the error reporting threshold to 255
8:15	Error Rate Degraded Threshold Trigger	0xFF	These bits provide the threshold value for reporting an error condition due to a degrading link. 0x00 - Disable the error rate register 0x01 - Set the error reporting threshold to 1 0x02 - Set the error reporting threshold to 2 ... 0xFF - Set the error reporting threshold to 255
16:31			Reserved

32.6.6 RLL Extended CAR/CSR Descriptions

The following sections provide detailed descriptions of the RLL extended CAR/CSRs.

32.6.6.1 RLL ECAR Control CSR (SRCFG0_ECARCTL)

This register provides the major control/configuration parameters for the RLL IP block. It should be considered "Static" from a software point of view. As such, it should only be written during system initialization or when the RLL and attached SRIO link have been quiesced and an idle condition is guaranteed.

Figure 32-79.RLL ECAR Control CSR (SRCFG0_ECARCTL)

Bit	Field Name	Reset State	Function
0	Promiscuous Mode	0b0	Set RLL to Promiscuous mode, all R2B packets will be accepted regardless of DestID value. If cleared, only packets with DestID matching RLL DevID value will be accepted.
1	Enable Soft Reset	0b0	Enable RLL logic to support soft reset functionality on inbound port/link reset requests from SRIO and user out of band reset (rll_bil_rst_req, input to BIL). 0 User Reset and SRIO reset request cause hard reset 1 User Reset and SRIO reset request cause soft reset
2:7			Reserved
8	Enable R2B Doorbell Retry	0b0	Enables automatic SRIO RETRY response generation on incoming R2B Doorbell request to busy Doorbell hardware.
9	Enable R2B Doorbell Error	0b0	Enables automatic SRIO ERROR response generation on incoming R2B Doorbell request to busy Doorbell hardware.
10	Enable R2B Message Retry	0b0	Enables automatic SRIO RETRY response generation on incoming R2B Message request to busy MBX/LTR slot hardware (see RLL_IP_messaging overview for implementation details).
11	Enable R2B Message Error	0b0	Enables automatic SRIO ERROR response generation on incoming R2B Message request to busy MBX/LTR slot hardware.

Figure 32-79.RLL ECAR Control CSR (SRCFG0_ECARCTL) (Continued)

12:29			Reserved
30:31	rio_ext_disc[1:0]	0b00	Current values of rio_ext_disc[1:0], as initialized via RIO_DEFS.dat. Read only

32.6.6.2 RLL ECAR R2B Mailbox Pointer 0 (Hi_Add) CSR (SRCFG0_ECARR2BMBPTR0H)

This CAR contains the upper address redirection pointer (in 66 bit address mode) for the Inbound R2B mailbox 0 functionality. This is available for use only in phase 1B of the RLL IP.

Figure 32-80.RLL ECAR R2B Mbox Pointer 0 (Hi_Add) CSR (SRCFG0_ECARR2BMBPTR0H)

Bit	Field Name	Reset State	Function
0:31	R2BMBX0 Ptr_Hi	0x0000	R2B Mailbox 0 redirection pointer upper address portion. Only utilized if 66-bit Addressing enabled. This register is utilized to form most significant address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A65,..., Bit31->A34

32.6.6.3 RLL ECAR R2B Mailbox Pointer 0 (Lo_Add) CSR (SRCFG0_ECARR2BMBPTR0L)

This CAR contains the lower address redirection pointer (in 66- or 34-bit address mode) for the Inbound R2B mailbox 0 functionality.

Figure 32-81.RLL ECAR R2B Mbox Pointer 0 (Lo_Add) CSR (SRCFG0_ECARR2BMBPTR0H)

Bit	Field Name	Reset State	Function
0:19	R2BMBX0 Ptr_Lo	0x00000	R2B Mailbox 0 redirection base pointer. Utilized to form upper address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A33,..., Bit19->A14.
20:31			Reserved

32.6.6.4 RLL ECAR R2B Mailbox Pointer 1 (Hi_Add) CSR (SRCFG0_ECARR2BMBPTR1H)

This CAR contains the upper address redirection pointer (in 66-bit address mode) for the Inbound R2B mailbox 1 functionality. This is available for use only in phase 1B of the RLL IP.

Figure 32-82.RLL ECAR R2B Mbox Pointer 1 (Hi_Add) CSR (SRCFG0_ECARR2BMBPTR1H)

Bit	Field Name	Reset State	Function
0:31	R2BMBX2 Ptr_Hi	0x0000	R2B Mailbox 1 redirection pointer upper address portion. Only utilized if 66 bit Addressing enabled. This register is utilized to form most significant address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A65,..., Bit31->A34.

32.6.6.5 RLL ECAR R2B Mailbox Pointer 1 (Lo_Add) CSR (SRCFG0_ECARR2BMBPTR1L)

This CAR contains the lower address redirection pointer (in 66- or 34-bit address mode) for the Inbound R2B mailbox 1 functionality.

Figure 32-83.RLL ECAR R2B Mbox Pointer 1 (Lo_Add) CSR (SRCFG0_ECARR2BMBPTR1L)

Bit	Field Name	Reset State	Function
0:19	R2BMBX2 Ptr_Lo	0x0000	R2B Mailbox 1 redirection base pointer. Utilized to form upper address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A33,..., Bit19->A14.
20:31			Reserved

32.6.6.6 RLL ECAR R2B Mailbox Pointer 2 (Hi_Add) CSR (SRCFG0_ECARR2BMBPTR2H)

This CAR contains the upper address redirection pointer (in 66-bit address mode) for the Inbound R2B mailbox 2 functionality. This is available for use only in phase 1B of the RLL IP.

Figure 32-84.RLL ECAR R2B Mbox Pointer 2 (Hi_Add) CSR (SRCFG0_ECARR2BMBPTR2H)

Bit	Field Name	Reset State	Function
0:31	R2BMBX2 Ptr_Hi	0x0000	R2B Mailbox 2 redirection pointer upper address portion. Only utilized if 66 bit Addressing enabled. This register is utilized to form most significant address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A65,..., Bit31->A34

32.6.6.7 RLL ECAR R2B Mailbox Pointer 2 (Lo_Add) CSR (SRCFG0_ECARR2BMBPTR2L)

This CAR contains the lower address redirection pointer (in 66- or 34-bit address mode) for the Inbound R2B mailbox 2 functionality.

Figure 32-85.RLL ECAR R2B Mbox Pointer 2 (Lo_Add) CSR (SRCFG0_ECARR2BMBPTR2L)

Bit	Field Name	Reset State	Function
0:19	R2BMBX2 Ptr_Lo	0x0000	R2B Mailbox 2 redirection base pointer. Utilized to form upper address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A33,..., Bit19->A14.
20:31			Reserved

32.6.6.8 RLL ECAR R2B Mailbox Pointer 3 (Hi_Add) CSR (SRCFG0_ECARR2BMBPTR3H)

This CAR contains the upper address redirection pointer (in 66-bit address mode) for the Inbound R2B mailbox 3 functionality. This is available for use only in phase 1B of the RLL IP.

Figure 32-86.RLL ECAR R2B Mbox Pointer 3 (Hi_Add) CSR

Bit	Field Name	Reset State	Function
0:31	R2BMBX3 Ptr_Hi	0x0000	R2B Mailbox 3 redirection pointer upper address portion. Only utilized if 66 bit Addressing enabled. This register is utilized to form most significant address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A65,..., Bit31->A34

32.6.6.9 RLL ECAR R2B Mailbox Pointer 3 (Lo_Add) CSR (SRCFG0_ECARR2BMBPTR3L)

This CAR contains the lower address redirection pointer (in 66- or 34-bit address mode) for the Inbound R2B mailbox 3 functionality.

Figure 32-87.RLL ECAR R2B Mbox Pointer 3 (Lo_Add) CSR

Bit	Field Name	Reset State	Function
0:19	R2BMBX3 Ptr_Lo	0x0000	R2B Mailbox 3 redirection base pointer. Utilized to form upper address bits of MWR_R spawned by incoming R2B MBX op. Bit0 -> A33,..., Bit19->A14.
20:31			Reserved

32.6.6.10 RLL ECAR R2B Doorbell Info CSR (SRCFG0_ECARDBLINFO)

This register contains the Doorbell information payload bytes following a R2B doorbell operation.

Figure 32-88.RLL ECAR R2B Doorbell Info CSR (SRCFG0_ECARDBLINFO)

Bit	Field Name	Reset State	Function
0:7	DBELL Info Lo	0x00	R2B Doorbell Info Byte (LSB).
8:15	DBELL Info Hi	0x00	R2B Doorbell Info Byte (MSB).
16:31			Reserved

32.6.6.11 RLL ECAR R2B Doorbell SRC CSR (SRCFG0_ECARDBLSRC)

This register contains the Doorbell Source ID and ID model size following a R2B doorbell operation.

Figure 32-89.RLL ECAR R2B Doorbell Info CSR (SRCFG0_ECARDBLSRC)

Bit	Field Name	Reset State	Function
0	DBELL R16	0b0	R2B Doorbell Source ID size model: 0b0 - Source ID 8 bits (bits 24-31 = SrcID) 0b1 - Source ID 16 bits (bits 16-31 = SrcID)
1:15			Reserved
16:31	DBELL Source ID	0x0000	Source ID of sender of this trapped Doorbell operation.

32.6.6.12 RLL ECAR B2R Address Window Base CSR 0 (SRCFG0_ECARB2RADDWINBAS0)

This register contains the base comparison address (least significant bits utilized in 66- or 34-bit addressing mode) of the Inbound B2R RLL address error checking window. This feature is only available in the RLL phase 1B IP.

Figure 32-90.RLL ECAR B2R Add Window Base CSR 0 (SRCFG0_ECARB2RADDWINBAS0)

Bit	Field Name	Reset State	Function
0:19	B2R Addr Base Lo	0x00000	B2R Addr Base Lo pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit19->A14 (FINAL TBD).
20:31			Reserved

32.6.6.13 RLL ECAR B2R Address Window Base CSR 1 (SRCFG0_ECARB2RADDWINBAS1)

This register contains the base comparison address (most significant bits utilized in 66-bit addressing mode) of the Inbound B2R RLL address error checking window. This feature is only available in the RLL phase 1B IP.

Figure 32-91.RLL ECAR B2R Add Window Base CSR 1 (SRCFG0_ECARB2RADDWINBAS1)

Bit	Field Name	Reset State	Function
0-19	B2R Addr Base Hi	0x00000000	B2R Addr Base Hi pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit31->A33 (FINAL TBD).

32.6.6.14 RLL ECAR B2R Address Window Limit CSR 0 (SRCFG0_ECARB2RADDWINLIM0)

This register contains the limit comparison address (least significant bits utilized in 66- or 34-bit addressing mode) of the Outbound B2R RLL address error checking window. This feature is only available in the RLL phase 1B IP.

Figure 32-92.RLL ECAR B2R Add Window Limit Base CSR 0 (SRCFG0_ECARB2RADDWINLIM0)

Bit	Field Name	Reset State	Function
0:19	B2R Addr Limit Lo	0xFFFFF	B2R Addr Limit Lo pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit19->A14 (FINAL TBD).
20:31			Reserved

32.6.6.15 RLL ECAR B2R Address Window Limit CSR 1 (SRCFG0_ECARB2RADDWINLIM1)

This register contains the limit comparison address (most significant bits utilized in 66-bit addressing mode) of the Inbound B2R RLL address error checking window. This feature is only available in the RLL phase 1B IP.

Figure 32-93.RLL ECAR B2R Add Window Base CSR 1 (SRCFG0_ECARB2RADDWINLIM1)

Bit	Field Name	Reset State	Function
0:19	B2R Addr Limit Hi	0xFFFFFFFF	B2R Addr Limit Hi pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit31->A33 (FINAL TBD).

32.6.6.16 RLL ECAR R2B Address Window Base CSR 0 (SRCFG0_ECARR2BADDWINBAS0)

This register contains the base comparison address (least significant bits utilized in 66- or 34-bit addressing mode) of the Inbound R2B RLL address error checking window. This feature is only available in the RLL phase 1B IP.

Figure 32-94.RLL ECAR R2B Add Window Base CSR 0 (SRCFG0_ECARR2BADDWINBAS0)

Bit	Field Name	Reset State	Function
0:19	R2B Addr Base Lo	0x00000	R2B Addr Base Lo pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit19->A14 (FINAL TBD).
20:31			Reserved

32.6.6.17 RLL ECAR R2B Address Window Base CSR 1 (SRCFG0_ECARR2BADDWINBAS1)

This register contains the base comparison address (most significant bits utilized in 66-bit addressing mode) of the Inbound R2B RLL address error checking window. This feature is only available in the RLL phase 1B IP.

Figure 32-95.RLL ECAR R2B Add Window Base CSR 1 (SRCFG0_ECARR2BADDWINBAS1)

Bit	Field Name	Reset State	Function
0-19	B2R Addr Base Hi	0x00000000	B2R Addr Base Hi pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit31->A33 (FINAL TBD).

32.6.6.18 RLL ECAR R2B Addr Window Base Limit CSR 0 (SRCFG0_ECARR2BADDWINLIM0)

This register contains the limit comparison address (least significant bits utilized in 66- or 34-bit addressing mode) of the Outbound R2B RLL address error checking window. This feature is only available in the RLL phase 1B IP

Figure 32-96.RLL ECAR R2B Add Window Limit CSR 0

Bit	Field Name	Reset State	Function
0:19	B2R Addr Limit Lo	0xFFFFF	B2R Addr Limit Lo pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit19->A14 (FINAL TBD).
20:31			Reserved

32.6.6.19 RLL ECAR R2B Addr Window Limit CSR 1 (SRCFG0_ECARR2BADDWINLIM1)

This register contains the limit comparison address (most significant bits utilized in 66-bit addressing mode) of the Inbound R2B RLL address error checking window. This feature is only available in the RLL phase 1B IP.

Figure 32-97.RLL ECAR R2B Add Window Limit CSR 1

Bit	Field Name	Reset State	Function
0:19	B2R Addr Limit Hi	0xFFFFFFFF	B2R Addr Limit Hi pointer. Utilized to validate B2R address from user. Bit0 -> A33,..., Bit31->A33 (FINAL TBD).

32.6.6.20 RLL ECAR BIL Event/Interrupt Status CSR (SRCFG0_ECARBILEVTINTSTAT)

This register contains the RLL Event/Interrupt (primary) status and clearing mechanisms. Bits in this CAR/CSR will be set in response to various conditions being detected by the RLL hardware. The event bits may (must) be cleared via user application/driver in order to detect future occurrences of the causal condition. If the corresponding Interrupt enable mask bits are set in the RLL ECAR BIL Interrupt Mask CSR, then the setting of one or more of these bits causes the assertion of the appropriate BIL interrupt signal to the user.

Figure 32-98.RLL ECAR BIL Event/Interrupt Status CSR

Bit	Field Name	Reset State	Function
0	Physical/Transport error occurred	0b0	This status bit indicates that one (or more) of the following bits in CSR 0x000158 is asserted: Output Retried-encountered - bit[11] Output Error-encountered - bit[14] Input Error-encountered - bit[22] Port Error - bit[29] The user must clear this bit to service the PT Event, and clear it's RCVD status. If the Physical/Transport error IRQ Enable is set in 0x1005c, the assertion of this bit will cause rll_bil_pterr_irq to be asserted, while this bit is asserted: R
1	RLL EME error occurred	0b0	Phase 1A - Always 1b0 Phase 1B IP Supported ONLY- A RLL EME (PhyTran) error event has occurred. R
2	RLL error occurred	0b0	Phase 1A - Always 1b0. Phase 1B IP Supported. ONLY- A RLL EME (Logical) error event has occurred. R
3	R2B Multicast CTL Sym Rcvd	0b0	A R2B Multicast Control Symbol has been received. The user must clear this bit to service the Multicast, and clear it's RCVD status.
4	R2B Port Write occurred	0b0	A R2B Port Write event has occurred, the port address, write size and SourceID have been captured, and the HW is busy, pending intervention by the user. The user must clear this bit to service the Port Write, and clear it's Occurred status. R/Clear
5	R2B Doorbell occurred	0b0	A R2B Doorbell event has occurred, the INFO data and SourceID have been captured, and the HW is busy, pending intervention by the user. The user must clear this bit to service the Doorbell, and clear it's Occurred status. R/Clear
6:11			Reserved
12	R2B Doorbell Discarded	0b0	A R2B Doorbell has been received and discarded, while the Doorbell HW was busy and the Enable R2B Doorbell Discard Trap bit was set in the ECAR Control CSR. R/Clear
13:15			Reserved
16	R2B Mailbox #0-letter 1 msg rcvd	0b0	All components of an Inbound Message to mailbox 0, letter slot 1 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
17	R2B Mailbox #0-letter 2 msg rcvd	0b0	All components of an Inbound Message to mailbox 0, letter slot 2 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
18	R2B Mailbox #0-letter 3 msg rcvd	0b0	All components of an Inbound Message to mailbox 0, letter slot 3 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
19	R2B Mailbox #0-letter 4 msg rcvd	0b0	All components of an Inbound Message to mailbox 0, letter slot 4 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
20	R2B Mailbox #1-letter 1 msg rcvd	0b0	All components of an Inbound Message to mailbox 1, letter slot 1 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear

Figure 32-98.RLL ECAR BIL Event/Interrupt Status CSR

21	R2B Mailbox #1-letter 2 msg rcvd	0b0	All components of an Inbound Message to mailbox 1, letter slot 2 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
22	R2B Mailbox #1-letter 3 msg rcvd	0b0	All components of an Inbound Message to mailbox 1, letter slot 3 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
23	R2B Mailbox #1-letter 4 msg rcvd	0b0	All components of an Inbound Message to mailbox 1, letter slot 4 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
24	R2B Mailbox #2-letter 1 msg rcvd	0b0	All components of an Inbound Message to mailbox 2, letter slot 1 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
25	R2B Mailbox #2-letter 2 msg rcvd	0b0	All components of an Inbound Message to mailbox 2, letter slot 2 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
26	R2B Mailbox #2-letter 3 msg rcvd	0b0	All components of an Inbound Message to mailbox 2, letter slot 3 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
27	R2B Mailbox #2-letter 4 msg rcvd	0b0	All components of an Inbound Message to mailbox 2, letter slot 4 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
28	R2B Mailbox #3-letter 1 msg rcvd	0b0	All components of an Inbound Message to mailbox 3, letter slot 1 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
29	R2B Mailbox #3-letter 2 msg rcvd	0b0	All components of an Inbound Message to mailbox 3, letter slot 2 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
30	R2B Mailbox #3-letter 3 msg rcvd	0b0	All components of an Inbound Message to mailbox 3, letter slot 3 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear
31	R2B Mailbox #3-letter 4 msg rcvd	0b0	All components of an Inbound Message to mailbox 3, letter slot 4 have been received from SRIO and acknowledged by the user. This bit must be cleared by the user to re-enable this Mbx/LtrSlt for further message reception. R/Clear

32.6.6.21 RLL ECAR BIL Interrupt Mask CSR (SRCFG0_ECARBILINTMASK))

This register contains the RLL Event/Interrupt mask/enable control. Bits in this CAR/CSR allow the various event status bits in the RLL ECAR BIL Event/Interrupt Status CSR, to be promoted in the RLL hardware to cause the assertion of the physical interrupt signals provided to the user on the RLL BIL interface.

Figure 32-99.RLL ECAR BIL Interrupt Mask CSR (SRCFG0_ECARBILINTMASK)

Bit	Field Name	Reset State	Function
0	Physical/Transport error IRQ Enable	0b0	Enable the generation of a hardware IRQ on the detection of a Phy/Transport error condition. Enable allows the assertion of the following bits in CSR 0x000158 to cause the assertion of rll_bil_pterr_irq: Output Retried-encountered - bit[11] Output Error-encountered - bit[14] Input Error-encountered - bit[22] Port Error - bit[29]
1	RLL EME error IRQ Enable	0b0	Phase 1A - Must Always be 1b0 Phase 1B IP Supported ONLY- A RLL EME (PhyTran) error event will cause assertion of rll_bil_emeerr_irq. R/Clear
2	RLL error IRQ Enable	0b0	Phase 1A - Must Always be 1b0 Phase 1B IP Supported ONLY A RLL EME (Logical) error event will cause assertion of rll_bil_rllerr_irq.
3	R2B CTL Sym IRQ Enable	0b0	Enable interrupt on detection of a R2B Multicast Control Symbol event occurred. Allows assertion of rll_bil_mcst_irq.
4	RLL Port Write IRQ Enable	0b0	Enable interrupt on detection of a R2B Port Write event occurred. Allows assertion of rll_bil_prtwr_irq.
5	R2B Doorbell IRQ Enable	0b0	Enable interrupt on detection of a R2B Doorbell event occurred. Allows assertion of rll_bil_dbell_irq.
6:11			Reserved
12	Enable R2B Doorbell Discard Trap	0b0	Enable detection and trapping of a R2B Doorbell received and discarded, while the Doorbell HW was busy. Enable allows the R2B Doorbell Discarded bit to be set in the ECAR Event/Interrupt CSR.
13	Enable R2B Message Discard Trap	0b0	Enable detection and trapping of a R2B Message received and discarded, while a letter slot's HW was busy. Enable allows the appropriate R2B Mailbox #x-letter n msg discard bit to be set in the ECAR MBX/Letter Status CSR.
14:15			Reserved
16	R2B Mailbox #0 msg IRQ Enable	0b0	Enable interrupt on detection of All components of an Inbound Message to mailbox 0, letter slot 0-3 having been received from SRIO and acknowledged by the user. Allows assertion of rll_bil_mbx0_irq on: R2B Mailbox #0-letter 1 msg rcvd or R2B Mailbox #0-letter 2 msg rcvd or R2B Mailbox #0-letter 3 msg rcvd or R2B Mailbox #0-letter 4 msg rcvd
17	R2B Mailbox #1 msg IRQ Enable	0b0	Enable interrupt on detection of All components of an Inbound Message to mailbox 1, letter slot 0-3 having been received from SRIO and acknowledged by the user. Allows assertion of rll_bil_mbx1_irq on: R2B Mailbox #1-letter 1 msg rcvd or R2B Mailbox #1-letter 2 msg rcvd or R2B Mailbox #1-letter 3 msg rcvd or R2B Mailbox #1-letter 4 msg rcvd

Figure 32-99.RLL ECAR BIL Interrupt Mask CSR (SRCFG0_ECARBILINTMASK) (Continued)

18	R2B Mailbox #2 msg IRQ Enable	0b0	Enable interrupt on detection of All components of an Inbound Message to mailbox 2, letter slot 0-3 having been received from SRIO and acknowledged by the user. Allows assertion of rll_bil_mbx2_irq on: R2B Mailbox #2-letter 1 msg rcvd or R2B Mailbox #2-letter 2 msg rcvd or R2B Mailbox #2-letter 3 msg rcvd or R2B Mailbox #2-letter 4 msg rcvd
19	R2B Mailbox #3 msg IRQ Enable	0b0	Enable interrupt on detection of All components of an Inbound Message to mailbox 3, letter slot 0-3 having been received from SRIO and acknowledged by the user. Allows assertion of rll_bil_mbx3_irq on: R2B Mailbox #3-letter 1 msg rcvd or R2B Mailbox #3-letter 2 msg rcvd or R2B Mailbox #3-letter 3 msg rcvd or R2B Mailbox #3-letter 4 msg rcvd
20:31			Reserved

32.6.6.22 RLL ECAR BIL MBX/Letter Status CSR (SRCFG0_ECARBILMBXLRSTAT)

This register contains the RLL Mailbox status and clearing mechanisms. Bits in this CAR/CSR are set in response to various conditions being detected by the RLL hardware. The event bits must be cleared by means of the user application/driver in order to detect future occurrences of the causal condition.

Figure 32-100.RLL ECAR BIL MBX/Letter Status CSR (SRCFG0_ECARBILMBXLRSTAT)

Bit	Field Name	Reset State	Function
0	R2B Mailbox #0-letter 1 msg discard	0b0	A R2B Message Segment to Mbx 0, Slt 1 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
1	R2B Mailbox #0-letter 2 msg discard	0b0	A R2B Message Segment to Mbx 0, Slt 2 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
2	R2B Mailbox #0-letter 3 msg discard	0b0	A R2B Message Segment to Mbx 0, Slt 3 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
3	R2B Mailbox #0-letter 4 msg discard	0b0	A R2B Message Segment to Mbx 0, Slt 4 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
4	R2B Mailbox #1-letter 1 msg discard	0b0	A R2B Message Segment to Mbx 1, Slt 1 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard Trap bit was set in the ECAR Control CSR.
5	R2B Mailbox #1-letter 2 msg discard	0b0	A R2B Message Segment to Mbx 1, Slt 2 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
6	R2B Mailbox #1-letter 3 msg discard	0b0	A R2B Message Segment to Mbx 1, Slt 3 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
7	R2B Mailbox #1-letter 4 msg discard	0b0	A R2B Message Segment to Mbx 1, Slt 4 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear

Figure 32-100.RLL ECAR BIL MBX/Letter Status CSR (SRCFG0_ECARBILMBXLRSTAT) (Continued)

8	R2B Mailbox #2-letter 1 msg discard	0b0	A R2B Message Segment to Mbx 2, Slt 1 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard Trap bit was set in the ECAR Control CSR. R/Clear
9	R2B Mailbox #2-letter 2 msg discard	0b0	A R2B Message Segment to Mbx 2, Slt 2 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
10	R2B Mailbox #2-letter 3 msg discard	0b0	A R2B Message Segment to Mbx 2, Slt 3 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
11	R2B Mailbox #2-letter 4 msg discard	0b0	A R2B Message Segment to Mbx 2, Slt 4 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
12	R2B Mailbox #3-letter 1 msg discard	0b0	A R2B Message Segment to Mbx 3, Slt 1 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
13	R2B Mailbox #3-letter 2 msg discard	0b0	A R2B Message Segment to Mbx 3, Slt 2 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
14	R2B Mailbox #3-letter 3 msg discard	0b0	A R2B Message Segment to Mbx 3, Slt 3 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard Trap bit was set in the ECAR Control CSR. R/Clear
15	R2B Mailbox #3-letter 4 msg discard	0b0	A R2B Message Segment to Mbx 3, Slt 4 has been received and discarded, while the Message HW was busy and the Enable R2B Message Discard bit was set in the ECAR Control CSR. R/Clear
16	R2B Mailbox #0-letter 1 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
17	R2B Mailbox #0-letter 2 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
18	R2B Mailbox #0-letter 3 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
19	R2B Mailbox #0-letter 4 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
20	R2B Mailbox #1-letter 1 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
21	R2B Mailbox #1-letter 2 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R

Figure 32-100.RLL ECAR BIL MBX/Letter Status CSR (SRCFG0_ECARBILMBXLRSTAT) (Continued)

22	R2B Mailbox #1-letter 3 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
23	R2B Mailbox #1-letter 4 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
24	R2B Mailbox #2-letter 1 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
25	R2B Mailbox #2-letter 2 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
26	R2B Mailbox #2-letter 3 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
27	R2B Mailbox #2-letter 4 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
28	R2B Mailbox #3-letter 1 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
29	R2B Mailbox #3-letter 2 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
30	R2B Mailbox #3-letter 3 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R
31	R2B Mailbox #3-letter 4 msg inproc	0b0	Current Status of R2B Mbx 0, Slt 1 Message HW. 0b0 = Idle 0b1 = In process of processing message Requests R

32.6.6.23 RLL ECAR Port Write Para Cap CSR (SRCFG0_ECARR2BPOTWRPRMCAP)

This register contains transfer parameter information specific to the occurrence of a R2B Port Write. It can be utilized to decode where and what another master actually did to the RLL CAR/CSR space, following the detection of a R2B Port Write Operation. This register *does not lock*. The contents always reflects the status of the last executed port write operation.

Figure 32-101.RLL ECAR R2B Port Write Para Cap CSR (SRCFG0_ECARR2BPOTWRPRMCAP)

Bit	Field Name	Reset State	Function
0:10			Reserved
11:15	R2B Port Wr Length	0b00000	Captured Port Write Length in 32-bit units.
16:31	R2B Port Wr Offset	0x0000	Starting offset (Mod 32 address) of this trapped R2B Port Write operation.

32.6.6.24 RLL ECAR Port Write Source Capture CSR (SRCFG0_ECARR2BPOTWRSRCCAP)

This register contains transfer parameter information specific to the occurrence of a R2B Port Write. It may be utilized to decode where and what another master actually did to the RLL CAR/CSR space, following the detection of a R2B Port Write Operation. This register *does not lock*. The contents will always reflect the status of the last executed port write operation.

Figure 32-102.RLL ECAR R2B Port Write Source Cap CSR (SRCFG0_ECARR2BPOTWRSRCCAP)

Bit	Field Name	Reset State	Function
0	R2B Port Wr R16	0b0	Captured Port Write R16 value, defining size of Source ID from last R2B Port Write operation. 0b0 -> Source ID 8 bits (bits 24-31 = SrcID) 0b1 -> Source ID 16 bits (bits 16-31 = SrcID)
1:5			Reserved
16:31	R2B Port Wr Source ID	0x0000	Source ID of sender of this trapped R2B Port Write operation.

32.6.6.25 RLL ECAR B2R Control Symbol CSR (SRCFG0_ECARB2RCTLSYM)

This CAR/CSR allows the generation of a B2R Multicast control symbol out to the SRIO fabric. Generally, Multicast control symbols are utilized for Time Of Day and or other synchronization functions within a SRIO system. Note: due to the buffered nature of Maint/Config operations within the RLL, delivery time from a CfgWR command being sent from the user through the B2R BIL to this Car, to the actual generation of a Multicast Control symbol to the fabric is not guaranteed.

Figure 32-103.RLL ECAR B2R Control Symbol CSR (SRCFG0_ECARB2RCTLSYM)

Bit	Field Name	Reset State	Function
0	Snd_MCST	0b0	Setting this bit to 1 will cause a single MultiCast control symbol to be sent to SRIO in the B2R direction. This bit must be reset to 0, to re-enable the generation of another multicast control symbol.
1:31			Reserved

32.6.7 GPL2SRIO Outbound Base Registers

These registers enable translation of PLB addresses to SRIO address and Device ID for Outbound Memory Regions 1, 2, 3, and Message, Configuration, and Maintenance Regions.

32.6.7.1 Address Low for Outbound Memory Region 1 (SRCFG0_SOMOMR1LAL)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-104.Address Low for Outbound Memory Region 1 (SRCFG0_SOMOMR1LAL)

Bit	Field Name	Reset State	Function
0:4	LAL	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLOMR1) for transactions that fall within the BAR set up by SRGPL0_OMR1BAH,L.
5:31			Reserved

32.6.7.2 Address High for Outbound Memory Region 1 (SRCFG0_SOMOMR1LAH)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-105.Address High for Outbound Memory Region 1 (SRCFG0_SOMOMR1LAH)

Bit	Field Name	Reset State	Function
0:31	LAH	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLOMR1) for transactions that fall within the BAR set up by SRGPL0_OMR1BAH,L.

32.6.7.3 Address Low for Outbound Memory Region 2 (SRCFG0_SOMOMR2LAL)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-106.Address Low for Outbound Memory Region 1 (SRCFG0_SOMOMR2LAL)

Bit	Field Name	Reset State	Function
0:11	LAL	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLOMR2) for transactions that fall within the BAR set up by SRGPL0_OMR2BAH,L.
12:31			Reserved

32.6.7.4 Address High for Outbound Memory Region 2 (SRCFG0_SOMOMR2LAH)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-107.Address High for Outbound Memory Region 1 (SRCFG0_SOMOMR2LAH)

Bit	Field Name	Reset State	Function
0:31	LAH	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLOMR2) for transactions that fall within the BAR set up by SRGPL0_OMR2BAH,L.

32.6.7.5 Address Low for Outbound Memory Region 3 (SRCFG0_SOMOMR3LAL)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-108.Address Low for Outbound Memory Region 3 (SRCFG0_SOMOMR3LAL)

Bit	Field Name	Reset State	Function
0:24	LAL	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLOMR3) for transactions that fall within the BAR set up by SRGPL0_OMR1BAH,L.
25:31			Reserved

32.6.7.6 Address High for Outbound Memory Region 3 (SRCFG0_SOMOMR3LAH)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-109.Address High for Outbound Memory Region 3 (SRCFG0_SOMOMR3LAH)

Bit	Field Name	Reset State	Function
0:31	LAH	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLOMR3) for transactions that fall within the BAR set up by SRGPL0_OMR1BAH,L.

32.6.7.7 Address Low for Message Region (SRCFG0_SOMMSGLAL)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-110.SOMMSG Address Low for Message Region (SRCFG0_SOMMSGLAL)

Bit	Field Name	Reset State	Function
0:24	LAL	0	Used to translate the PLB address into a SRIO address and device ID (see SRCFG0_DIDLMSG) for transactions that fall within the BAR set up by SRGPL0_MSGBAH,L.
25:31			Reserved

32.6.7.8 Address High for Message Region (SRCFG0_SOMMSGLAH)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-111.Address High for Message Region (SRCFG0_SOMMSGLAH)

Bit	Field Name	Reset State	Function
0:31	LAH	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLMSG) for transactions that fall within the BAR set up by SRGPL0_MSGBAH,L.

32.6.7.9 Address Low for Configuration Region (SRCFG0_SOMCFGLAL)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-112.Address Low for Configuration Region (SRCFG0_SOMCFGLAL)

Bit	Field Name	Reset State	Function
0:24	LAL	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLCFG) for transactions that fall within the BAR set up by SRGPL0_CFGBAH,L.
25:31			Reserved

32.6.7.10 Address High for Configuration Region (SRCFG0_SOMCFGLAH)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-113.Address High for Configuration Region (SRCFG0_SOMCFGLAH)

Bit	Field Name	Reset State	Function
0:31	LAH	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLCFG) for transactions that fall within the BAR set up by SRGPL0_CFGBAH,L.

32.6.7.11 Address Low for Maintenance Region (SRCFG0_SOMMNTLAL)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-114.Address Low for Maintenance Region (SRCFG0_SOMMNTLAL)

Bit	Field Name	Reset State	Function
0:24	LAL	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLMNT) for transactions that fall within the BAR set up by SRGPL0_MNTBAH,L.
25:31			Reserved

32.6.7.12 Address High for Maintenance Region (SRCFG0_SOMMNTLAH)

This register enables translation of PLB addresses to SRIO address and Device ID.

Figure 32-115.Address High for Maintenance Region (SRCFG0_SOMMNTLAH)

Bit	Field Name	Reset State	Function
0:31	LAH	0	Used to translate the PLB address into a SRIO address and Device ID (see SRCFG0_DIDLMNT) for transactions that fall within the BAR set up by SRGPL0_MNTBAH,L.

32.6.8 GPL2SRIO Outbound Destination Location Registers

These registers contain the Destination ID Locations for Outbound Memory Regions 1, 2, 3, and Message, Configuration, and Maintenance Regions.

32.6.8.1 Destination ID Location for Outbound Memory Region 1 (SRCFG0_DIDLOMR1)

This register contains the Destination ID Location.

Figure 32-116.Destination ID Location for Outbound Memory Region 1 (SRCFG0_DIDLOMR1)

Bit	Field Name	Reset State	Function
0:25			Reserved
26:31	DIDL	0x1D	SRIO Outbound Destination ID Location for Memory Region 1 Describes where the least significant bit of the 8-bit Destination ID is located within the 64-bit Address (After the PLB Address has been translated using SRCFG0_SOMOMR1LAL,H). Valid values are decimal 29–60.

32.6.8.2 Destination ID Location for Outbound Memory Region 2 (SRCFG0_DIDLOMR2)

This register contains the Destination ID Location.

Figure 32-117.Destination ID Location for Outbound Memory Region 2 (SRCFG0_DIDLOMR2)

Bit	Field Name	Reset State	Function
0:25			Reserved
26:31	DIDL	0x1D	SRIO Outbound Destination ID Location for Memory Region 2 Describes where the least significant bit of the 8-bit Destination ID is located within the 64-bit Address (After the PLB Address has been translated using SRCFG0_SOMOMR2LAL,H). Valid values are decimal 29–60.

32.6.8.3 Destination ID Location for Outbound Memory Region 3 (SRCFG0_DIDLOMR3)

This register contains the Destination ID Location.

Figure 32-118.Destination ID Location for Outbound Memory Region (SRCFG0_DIDLOMR3)

Bit	Field Name	Reset State	Function
0:25			Reserved
26:31	DIDL	0x1D	SRIO Outbound Destination ID Location for Memory Region 3 Describes where the least significant bit of the 8-bit Destination ID is located within the 64-bit Address (After the PLB Address has been translated using SRCFG0_SOMOMR3LAL,H). Valid values are decimal 29–60.

32.6.8.4 Destination ID Location for Outbound Message Region (SRCFG0_DIDLMSG)

This register contains the Destination ID Location.

Figure 32-119.Destination ID Location for Outbound Message Region (SRCFG0_DIDLMSG)

Bit	Field Name	Reset State	Function
0:25			Reserved
26:31	DIDL	0x1D	SRIO Outbound Destination ID Location for Message Region Describes where the least significant bit of the 8-bit Destination ID is located within the 64-bit Address (After the PLB Address has been translated using SRCFG0_SOMMSGLAL,H). Valid values are decimal 29–60.

32.6.8.5 Destination ID Location for Maintenance Region (SRCFG0_DIDLMNT)

This register contains the Destination ID Location.

Figure 32-120.Destination ID Location for Maintenance Region (SRCFG0_DIDLMNT)

Bit	Field Name	Reset State	Function
0:25			Reserved
26:31	DIDL	0x1D	SRIO Outbound Destination ID Location for Maintenance Region Describes where the least significant bit of the 8-bit Destination ID is located within the 64-bit Address (After the PLB Address has been translated using SRCFG0_SOMMNTLAL,H). Valid values are decimal 29–60. Note, the SRIO address for Maintenance commands has the following meaning: SRIO address [31:24] = Hop count, SRIO address[23:3] = CAR/CSR address.

32.6.9 GPL2SRIO Inbound Base and Mask Registers

These registers enable translation of SRIO addresses to PLB addresses for the Inbound Memory Regions 1 and 2.

32.6.9.1 BAR0 Inbound Base Address Register 0 (SRCFG0_BAR0)

This register defines the SRIO Base Address 0 (BAR0 for inbound transactions).

Figure 32-121.BAR0 Inbound Base Address Register 0 (SRCFG0_BAR0)

Bit	Field Name	Reset State	Function
0:13	BAR	0	SRIO Base Address Maps to bits 0:13 of 34-bit SRIO address. A value of 0xC000_0000 in this register defines an SRIO base address of 0x3_0000_0000.
14:31			Reserved

32.6.9.2 BAR0M Inbound Base Address Register Mask 0 (SRCFG0_BAR0M)

This register enables and sets the size of the SRIO Base Address 0 (BAR0 for inbound transactions).

Figure 32-122.BAR0M Inbound Base Address Register Mask 0 (SRCFG0_BAR0M)

Bit	Field Name	Reset State	Function
0:13	BAR	0	SRIO Base Address Maps to bits 0:13 of 34-bit SRIO address. A value of 0xC000_0000 in this register defines an SRIO base address of 0x3_0000_0000.
14:30			Reserved
31	BEN	0	When set to 1, BAR0 is enabled.

32.6.9.3 SIM0 Inbound (SRCFG0_SIM01S)

This register enables customizing the SRIO address space SIM within the PLB address space.

Figure 32-123.SIM0 Inbound Size (SRCFG0_SIM01S)

Bit	Field Name	Reset State	Function
0:24	SIZ	0	SIM01 for BAR0 Specifies the offset that splits BAR0 into two regions. Transactions within BAR0 that are located below the offset indicated by this register are translated using SIM0. All other transactions are translated using SIM1. The value of the offset range from 1KB to the size of the first address region (16GB). Bits 0:23 of the offset that splits the 34-bit BAR0 memory region can be masked. The mask defines the bits that are considered by the address decoder, thus determining the value of the region offset. The offset value can range from 1KB: 0000000000.....0000000000 - 16GB 1000000000.....0000000000 - 8GB 1100000000.....0000000000 - 4GB --- 1111111111.....1111111100 - 4KB 1111111111.....1111111110 - 2KB 1111111111.....1111111111 - 1KB The offset value must be correlated with the mask input for the BAR0 so that the indicated offset value is not larger than the BAR0 size. Bits 0:24 map to bits 0:24 of a 34-bit size mask. A value of 0xF000_0000 in this register declares an offset of 1GB.
25:31			Reserved

32.6.9.4 SIM0 Address Low for Inbound BAR 0 (SRCFG0_SIM0LAL)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-124.SIM0 Address Low for Inbound (SRCFG0_SIM0LAL)

Bit	Field Name	Reset State	Function
0:21	LAL	0	SIM0 Address Low for BAR0 Used to translate the SRIO address into a PLB address for transactions that fall within BAR0 memory region and are located below the offset indicated by SRCFG0_SIM01S. Maps to bits 32:53 of the PLB address.
22:31			Reserved

32.6.9.5 SIM0 Address High for Inbound BAR 0 (SRCFG0_SIM0LAH)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-125.SIM0 Address High for Inbound (SRCFG0_SIM0LAH)

Bit	Field Name	Reset State	Function
0:31	LAL	0	SIM0 Address High for BAR0 Used to translate the SRIO address into a PLB address for transactions that fall within BAR0 memory region and are located below the offset indicated by SRCFG0_SIM01S. Maps to bits 0:31 of the PLB address.

32.6.9.6 SIM1 Address Low for Inbound BAR 0 (SRCFG0_SIM1LAL)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-126.SIM1 Address Low for Inbound BAR 0 (SRCFG0_SIM1LAL)

Bit	Field Name	Reset State	Function
0:11	LAL	0	SIM1 Address Low for BAR0 Used to translate the SRIO address into a PLB address for transactions that fall within BAR0 memory region and are located below the offset indicated by SRCFG0_SIM01S. Maps to bits 32:43 of the PLB address.
12:31			Reserved

32.6.9.7 SIM1 Address High for Inbound BAR 0 (SRCFG0_SIM1LAH)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-127.SIM1 Address High for Inbound BAR 0 (SRCFG0_SIM1LAH)

Bit	Field Name	Reset State	Function
0:31	LAL	0	SIM1 Address High for BAR0 Used to translate the SRIO address into a PLB address for transactions that fall within BAR0 memory region and are located below the offset indicated by SRCFG0_SIM01S. Maps to bits 0:31 of the PLB address.

32.6.9.8 BAR1 Inbound Base Address Register 1 (SRCFG0_BAR1)

This register defines the SRIO Base Address 1 (BAR1 for inbound transactions).

Figure 32-128.BAR1 Inbound Base Address Register 1 (SRCFG0_BAR1)

Bit	Field Name	Reset State	Function
0:13	BAR	0	SRIO Base Address Maps to bits 0:13 of 34-bit SRIO address. A value of 0xC000_0000 in this register defines an SRIO base address of 0x3_0000_0000.
14:31			Reserved

32.6.9.9 BAR1 Inbound Base Address Register Mask 1 (SRCFG0_BAR1M)

This register enables and sets the size of the SRIO Base Address 1 (BAR1 for inbound transactions).

Figure 32-129.BAR1 Inbound Base Address Register Mask 1 (SRCFG0_BAR1M)

Bit	Field Name	Reset State	Function
0:13	BAR	0	SRIO Base Address Maps to bits 0:13 of 34-bit SRIO address. A value of 0xC000_0000 in this register defines an SRIO base address of 0x3_0000_0000.
14:30			Reserved
31	BEN	0	When set to 1, BAR1 is enabled.

32.6.9.10 SIM23 Inbound (SRCFG0_SIM23S)

This register enables customizing the SRIO address space SIM within the PLB address space.

Figure 32-130.SIM23S Inbound (SRCFG0_SIM23S)

Bit	Field Name	Reset State	Function
0:24	SIZ	0	SIM23 for BAR1 Specifies the offset that splits BAR1 into two regions. Transactions within BAR0 that are located below the offset indicated by this register are translated using SIM2. All other transactions are translated using SIM3. The value of the offset range from 1KB to the size of the first address region (16GB). Bits 0:23 of the offset that splits the 34-bit BAR0 memory region can be masked. The mask defines the bits that are considered by the address decoder, thus determining the value of the region offset. The offset value can range from 1KB: 0000000000.....0000000000 - 16GB 1000000000.....0000000000 - 8GB 1100000000.....0000000000 - 4GB --- 1111111111.....1111111100 - 4KB 1111111111.....1111111110 - 2KB 1111111111.....1111111111 - 1KB The offset value must be correlated with the mask input for the BAR1 so that the indicated offset value is not larger than the BAR1 size. Bits 0:24 map to bits 0:24 of a 34-bit size mask. A value of 0xF000_0000 in this register declares an offset of 1GB.
25:31			Reserved

32.6.9.11 SIM2 Address Low for Inbound BAR 1 (SRCFG0_SIM2LAL)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-131.SIM2 Address Low for Inbound BAR 1 (SRCFG0_SIM2LAL)

Bit	Field Name	Reset State	Function
0:21	LAL	0	SIM0 Address Low for BAR1 Used to translate the SRIO address into a PLB address for transactions that fall within BAR1 memory region and are located below the offset indicated by SRCFG0_SIM23S. Maps to bits 32:53 of the PLB address.
22:31			Reserved

32.6.9.12 SIM2 Address High for Inbound BAR 1 (SRCFG0_SIM2LAH)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-132.SIM2 Address High for Inbound BAR 1 (SRCFG0_SIM2LAH)

Bit	Field Name	Reset State	Function
0:31	LAL	0	SIM0 Address High for BAR1 Used to translate the SRIO address into a PLB address for transactions that fall within BAR1 memory region and are located below the offset indicated by SRCFG0_SIM23S. Maps to bits 0:31 of the PLB address.

32.6.9.13 SIM3 Address Low for Inbound BAR 1 (SRCFG0_SIM3LAL)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-133.SIM3 Address Low for Inbound BAR 1 (SRCFG0_SIM3LAL)

Bit	Field Name	Reset State	Function
0:11	LAL	0	SIM1 Address Low for BAR1 Used to translate the SRIO address into a PLB address for transactions that fall within BAR1 memory region and are located below the offset indicated by SRCFG0_SIM23S. Maps to bits 32:43 of the PLB address.
12:31			Reserved

32.6.9.14 SIM3 Address High for Inbound BAR 1 (SRCFG0_SIM3LAH)

This register enables translation of SRIO addresses to PLB addresses.

Figure 32-134.SIM3 Address High for Inbound BAR 1 (SRCFG0_SIM3LAH)

Bit	Field Name	Reset State	Function
0:31	LAL	0	SIM1 Address High for BAR1 Used to translate the SRIO address into a PLB address for transactions that fall within BAR1 memory region and are located below the offset indicated by SRCFG0_SIM23S. Maps to bits 0:31 of the PLB address.

32.6.10 GPL2SRIO SRREG0 Registers

Writes to register offsets that are not defined in this table cause unexpected results. Writes to register offsets that are read-only cause unexpected results. Other registers in this region might be corrupted.

These registers are accessed by means of the BAR.

Table 32-10. SRIO SRREG0_xxxx Registers

Mnemonic	Description	Offset	Access	Page
General SRREG0 Registers				
SRREG0_BRCTL	Bridge Control	0x00000	R/W	1179
SRREG0_BRSTAT	Bridge Status	0x00008	R	1179
SRREG0_BRINT	Bridge Interrupt	0x00010	See register	1180
SRREG0_PRIO	Bridge Transaction Priority	0x00018	R/W	1181

Table 32-10. SRIO SRREG0_xxxx Registers (Continued)

Mnemonic	Description	Offset	Access	Page
SRREG0_ID1	SRIO Device ID, Device Vendor ID (for CARCSR Address 0)	0x00020	R/W	1181
SRREG0_ID2	Device Information CAR (for CARCSR Address 4)	0x00028	R/W	1181
SRREG0_ID3	Assembly ID CAR (for CARCSR Address 8)	0x00030	R/W	1181
SRREG0_ID4	Assembly Information CAR (for CARCSR Address C)	0x00038	R/W	1182
SRREG0_SMSTAT	State Machine Status	0x00080	R	1182
SRREG0_SP	Scratch pad	0x00090	R/W	1182
SRREG0_OATMDIN	Read data for outbound atomic commands	0x00098	R	1183
Mail Box Message Length and Segment Size Registers				
SRREG0_MSGLSEGS11	Message Length & Segment Size Register for MB 1 Letter 1	0x00100	R/W	1183
SRREG0_MSGLSEGS12	Message Length & Segment Size Register for MB 1 Letter 2	0x00108	R/W	1183
SRREG0_MSGLSEGS13	Message Length & Segment Size Register for MB 1 Letter 3	0x00110	R/W	1183
SRREG0_MSGLSEGS14	Message Length & Segment Size Register for MB 1 Letter 4	0x00118	R/W	1183
SRREG0_MSGLSEGS21	Message Length & Segment Size Register for MB 2 Letter 1	0x00120	R/W	1183
SRREG0_MSGLSEGS22	Message Length & Segment Size Register for MB 2 Letter 2	0x00128	R/W	1183
SRREG0_MSGLSEGS23	Message Length & Segment Size Register for MB 2 Letter 3	0x00130	R/W	1183
SRREG0_MSGLSEGS24	Message Length & Segment Size Register for MB 2 Letter 4	0x00138	R/W	1183
SRREG0_MSGLSEGS31	Message Length & Segment Size Register for MB 3 Letter 1	0x00140	R/W	1183
SRREG0_MSGLSEGS32	Message Length & Segment Size Register for MB 3 Letter 2	0x00148	R/W	1183
SRREG0_MSGLSEGS33	Message Length & Segment Size Register for MB 3 Letter 3	0x00150	R/W	1183
SRREG0_MSGLSEGS34	Message Length & Segment Size Register for MB 3 Letter 4	0x00158	R/W	1183
SRREG0_MSGLSEGS41	Message Length & Segment Size Register for MB 4 Letter 1	0x00160	R/W	1183
SRREG0_MSGLSEGS42	Message Length & Segment Size Register for MB 4 Letter 2	0x00168	R/W	1183
SRREG0_MSGLSEGS43	Message Length & Segment Size Register for MB 4 Letter 3	0x00170	R/W	1183
SRREG0_MSGLSEGS44	Message Length & Segment Size Register for MB 4 Letter 4	0x00178	R/W	1183
Outbound Maximum Request Size Register				
SRREG0_OBMAXS	Outbound Max Request Size	0x00200	R/W	1183
Door Bell Register				
SRREG0_DBLCTL	Door Bell Control	0x00300	R/W	1184
Transmit Control Register				
SRREG0_XFRCTL	Transfer Control	0x00400	R/W	1184
Outbound Atomic Registers				
SRREG0_OATMCTL	Outbound Atomic Control	0x00500	R/W	1184
SRREG0_OATMADR	Outbound Atomic Address	0x00508	R/W	1185
SRREG0_OATMDID	Outbound Atomic Destination ID	0x00510	R/W	1185
SRREG0_OATMDOUT	Outbound Atomic Data Out Register for Test and Swap	0x00520	R/W	1185
Local Atomic Registers				
SRREG0_LATMCTL	Local Atomic Control	0x00528	See reg	1185
SRREG0_LATMADR	Local Atomic SRIO Address Space	0x00530	R/W	1186
SRREG0_LATMDOUT	Local Atomic Data Out for certain Local Atomic commands	0x00540	R/W	1186
SRREG0_LATMDIN	Local Atomic Data In for certain Local Atomic commands	0x00548	R	1186
Inbound Transaction Timeout Register				
SRREG0_INBTO	Inbound Transaction Timeout	0x00600	R/W	1186
SRIO Response Completion Register				
SRREG0_RSPSTAT	SRIO Response Completion Status	0x00700	R	1187

The following sections provide detailed descriptions of the SRREG0 registers.

32.6.10.1 Bridge Control Register (SRREG0_BRCTL)

This register provides control functions for the GBIF2RIO Bridge.

Figure 32-135.Bridge Control Register (SRREG0_BRCTL)

Bit	Field Name	Reset State	Function
0:18			Reserved
19	SRIM	0	Inbound interface received a valid SRIO reset sequence Mask
20	OCSM	0	Outbound command with non-data Response Completion Interrupt Mask
21			Reserved
22	ITTOM	0	Inbound Transaction Timeout Mask
23	LAWER	0	Local Atomic Write Error Interrupt Mask
24	LAREM	0	Local Atomic Read Error Interrupt Mask
25			Reserved
26	HGPT	0	Hold all pending transactions Only ongoing transactions will be completed. Once this bit is set, it can only be cleared when the entire block is reset.
27	FGGB	0	Assertion causes an invalidation of the general-purpose buffer handler. Keep asserted until the fggba bit asserts in the SRREG0_BRSTAT register.
28	SINGRE	0	Force single inbound read outstanding This bit must be set to 1 for proper operation.
29	BILRST	0	Request that the RLL logic block be reset
30	B2RRST	0	Request that a link request reset sequence be sent out the SRIO port
31			Reserved

32.6.10.2 Bridge Status Register (SRREG0_BRSTAT)

This register provides status information for the GBIF2RIO Bridge.

.

Figure 32-136.Bridge Status Register (SRREG0_BRSTAT)

Bit	Field Name	Reset State	Function
0:28			Reserved
29	RDEVRS	0	RLL received a link request reset sequence All RLL internal logic is currently being reset.
30	PLBID	0	PLB logic is in idle state and can be reset
31	FGGBA	0	Acknowledges the invalidating of the general-purpose buffer

32.6.10.3 Bridge Interrupt Register (SRREG0_BRINT)

This register provides interrupt control and status information for the GBIF2RIO Bridge.

Figure 32-137.Bridge Interrupt Register (SRREG0_BRINT)

Bit	Field Name	Reset State	Function
0	RBPTER	0	SRIO Phy-Transport Base Level Error R
1	RBEMME	0	SRIO EME Logical Layer Error R
2	RBRLLE	0	RLL Hardware Layer Error R
3	RBMCST	0	Multicast Event control symbol was received from the SRIO R
4	RLBPRT	0	SRIO Port Write to internal CAR/CSR Space occurred. R
5	RLDBEL	0	Doorbell was received from the SRIO R
6:15			Reserved
16	RBMX0	0	Mailbox 0 complete message was received from the SRIO R
17	RBMX1	0	Mailbox 1 complete message was received from the SRIO R
18	RBMX2	0	Mailbox 2 complete message was received from the SRIO R
19	RBMX3	0	Mailbox 3 complete message was received from the SRIO R
20:23			Reserved
24	SRINT	0	RIO inbound interface received a valid RIO reset sequence Interrupt R/C
25	OCSTA	0	Outbound command with non-data Response Completion Interrupt Status found in SRREG0_RSPSTAT R/C
26			Reserved
27	ITTO	0	Inbound Transaction Timeout R/C
28	LARW	0	Local Atomic Write Error Interrupt R/C
29	LARE	0	Local Atomic Read Error Interrupt R/C
30	BRINTE	0	SRIO Bridge Interrupts Enabled R/W
31	RLLIEN	0	SRIO RLL Interrupts Enabled R/W

32.6.10.4 Transaction Priority Register (SRREG0_PRIO)

The two-bit SRIO priority for each outbound command type is set in this register. Zero is lowest priority, three is highest. To avoid system deadlock, do not set these request priority levels to the highest priority value of three.

Figure 32-138.Transaction Priority Register (SRREG0_PRIO)

Bit	Field Name	Reset State	Function
0:15			Reserved
16:17	OAPRI	0	Outbound Atomic Priority
18:19	ODBELP	0	Outbound Doorbell Priority
20:21	OMAREP	0	Outbound Maintenance Region Priority
22:23	OCRP	0	Outbound Configuration Region Priority
24:25	OMRP	0	Outbound Mailbox Region Priority
26:27	OMR3P	0	Outbound Memory Region 3 Priority
28:29	OMR2P	0	Outbound Memory Region 2 Priority
30:31	OMR1P	0	Outbound Memory Region 1 Priority

32.6.10.5 ID Register 1 (SRREG0_ID1)

This register populates the Device ID information that is read from the SRCFG0_xxxx register (CARCSR address 0).

Figure 32-139.ID Register 1 (SRREG0_ID1)

Bit	Field Name	Reset State	Function
0:15	SDID	0x0100	Device ID
16:31	SVID	0x0064	Device Vendor ID

32.6.10.6 ID Register 2 (SRREG0_ID2)

This register populates the Device information that is read from the SRCFG0_xxxx register (CARCSR address 4).

Figure 32-140.ID Register 2 (SRREG0_ID2)

Bit	Field Name	Reset State	Function
0:15			Reserved
16:23	DREV	0x00	Device revision level
24:31	SREV	0x12	Specification Revision Level Compliance (1.2)

32.6.10.7 ID Register 3 (SRREG0_ID3)

This register populates the Assembly ID information that is read from the SRCFG0_xxxx register (CARCSR address 8).

Figure 32-141.ID Register 3 (SRREG0_ID3)

Bit	Field Name	Reset State	Function
0:15	AID	0	Assembly ID
16:31	AVID	0	Assembly Vendor ID

32.6.10.8 ID Register 4 (SRREG0_ID4)

This register populates the assembly revision information that is read from the SRCFG0_xxxx register (CARCSR address C). However, the extended features pointer read from SRCFG0_xxxx is always 0x0100.

Figure 32-142.ID Register 4 (SRREG0_ID4)

Bit	Field Name	Reset State	Function
0:15	AREV	0	Assembly Revision Information Assembly Revision Extended features pointer always reads 0x0100.
16:31			Reserved

32.6.10.9 State Machine Status Register (SRREG0_SMSTAT)

This register provides the current state information for the Inbound and Outbound Request state machines.

Figure 32-143.State Machine Status Register (SRREG0_SMSTAT)

Bit	Field Name	Reset State	Function
0:4			Reserved
5:15	ORCSC	0x0001	Outbound Request State Definitions 0x0001 OUT_REQ_CTRL_SM_IDLE 0x0002 OUT_REQ_CTRL_SM_REQ 0x0004 OUT_REQ_CTRL_SM_RESP 0x0008 OUT_REQ_CTRL_SM_MEMWR 0x0010 OUT_REQ_CTRL_SM_MEMRD 0x0020 OUT_REQ_CTRL_SM_WR_XFR1 0x0040 OUT_REQ_CTRL_SM_WR_XFR2 0x0080 OUT_REQ_CTRL_SM_CLEAN_UP 0x0100 OUT_REQ_CTRL_SM_WAIT 0x0200 OUT_REQ_CTRL_SM_DBELL 0x0400 OUT_REQ_CTRL_SM_ATOMIC
16:31	IRCSC	0x0001	Inbound Request State Definitions 0x0001 IN_REQ_CTRL_SM_IDLE 0x0002 IN_REQ_CTRL_SM_REQ 0x0004 IN_REQ_CTRL_SM_RESP 0x0008 IN_REQ_CTRL_SM_MEMWR 0x0010 IN_REQ_CTRL_SM_MEMRD 0x0020 IN_REQ_CTRL_SM_MEMRD_ACK 0x0040 IN_REQ_CTRL_SM_WR_XFR1 0x0080 IN_REQ_CTRL_SM_WR_XFR2 0x0100 IN_REQ_CTRL_SM_XFR_GBIF 0x0200 IN_REQ_CTRL_SM_CLEAN_UP 0x0400 IN_REQ_CTRL_SM_WAIT 0x0800 IN_REQ_CTRL_SM_ATOMICRD 0x1000 IN_REQ_CTRL_SM_ATOMICRD_ACK 0x2000 IN_REQ_CTRL_SM_ATOMICRDXFR 0x3000 IN_REQ_CTRL_SM_ATOMICWRCAL 0x4000 IN_REQ_CTRL_SM_ATOMICWR

32.6.10.10 Scratch Pad Register (SRREG0_SP)

This register can be used as a scratch pad by software.

Figure 32-144.Scratch Pad Register (SRREG0_SP)

Bit	Field Name	Reset State	Function
0:31			Reserved

32.6.10.11 Outbound Atomic Data In Register (SRREG0_OATMDIN)

This register contains the read data for Outbound Atomic transactions.

Figure 32-145.Outbound Atomic Data In Register (SRREG0_OATMDIN)

Bit	Field Name	Reset State	Function
0:31	ADIN	0	Read data returned from Outbound Atomic commands

32.6.10.12 Message Length and Segment Size Registers (SRREG0_MSGLSEGSxx)

These registers define the message length and segment size for the Mail boxes. This register description is used for registers SRREG0_MSGLSEGS11–SRREG0_MSGLSEGS44.

Figure 32-146.Message Length and Segment Size Registers (SRREG0_MSGLSEGSxx)

Bit	Field Name	Reset State	Function
0:23			Reserved
24:27	MSGLEN	0	Mail box Message Length in number of segments A value of 0x0 means one segment, and a value of 0xF means 16 segments.
28:31	SEGSIZ	0	Mail box Segment Size 1011 8B/msg_seg–segsize 1010 16B/msg_seg– segsize 1011 32B/msg_seg–segsize 1100 64B/msg_seg– segsize 1101 128B/msg_seg–segsize 1110 256B/msg_seg–segsize

32.6.10.13 Outbound Maximum Request Size Register (SRREG0_OBMAXS)

This register controls the PLB maximum request size to the SRIO logic. This register should not be changed from the default value.

Figure 32-147.Outbound Maximum Request Size Register (SRREG0_OBMAXS)

Bit	Field Name	Reset State	Function
0:25			Reserved
26:28	ORMRSZ	1	Outbound Read Max Request Size 000 128B 001 256B 010 512B 011 1024B 100 2048B 101 4096B 110 Reserved 111 Reserved
29:31	OWMRSZ	1	Outbound Write Max Request Size 000 128B 001 256B 010 512B 011 1024B 100 2048B 101 4096B 110 Reserved 111 Reserved

32.6.10.14 Doorbell Control Register (SRREG0_DBLCTL)

This register is used to generate Outbound Doorbell transactions.

Figure 32-148.Doorbell Control Register (SRREG0_DBLCTL)

Bit	Field Name	Reset State	Function
0:7	ODBDID	0	Outbound Doorbell Destination Id
8:23	DBMSG	0	Doorbell Message
24:30			Reserved
31	ODBEE	0	Outbound Doorbell Event Enable Writing this bit causes the Bridge to perform a DBEL_R transaction. This bit is automatically cleared by the bridge when the command is issued.

32.6.10.15 Flow Control Register (SRREG0_XFRCTL)

This register can be used to provide flow control for RLL transmit operations.

Figure 32-149.Flow Control Register (SRREG0_XFRCTL)

Bit	Field Name	Reset State	Function
0:30			Reserved
31	SB2XOF	0	RLL B2R Xoff Suspends RLL transmit operations. **Note:** There is no support for sending or receiving type 7 packets.

32.6.10.16 Outbound Atomic Control Register (SRREG0_OATMCTL)

This register is used to generate Outbound Atomic transactions.

Figure 32-150.Outbound Atomic Control Register (SRREG0_OATMCTL)

Bit	Field Name	Reset State	Function
0	OAEVEN	0	Outbound Atomic Event Enable Writing this bit causes the Bridge to perform one of five possible Atomic transactions. This bit is automatically cleared by the bridge when the command is issued.
1:23			Reserved
24:27	OASIZE	0	Outbound Atomic Valid Bytes 0000 Byte 0 Valid 0001 Byte 1 Valid 0010 Byte 2 Valid 0011 Byte 3 Valid 0100 Bytes 0-1 Valid 0110 Bytes 2-3 Valid 1000 Bytes 0-3 Valid
28:31	OATRTY	0	Outbound Atomic Transaction Type 1000 ATMTS_R - Atomic Test and Swap with Response 1010 ATMINC_R - Atomic Increment with Response 1011 ATMDEC_R - Atomic Decrement with Response 1100 ATMSET_R - Atomic Set with Response 1111 ATMCLR_R - Atomic Clear with Response

32.6.10.17 Outbound Atomic Address Register (SRREG0_OATMADR)

This register contains the SRIO address for Outbound Atomic transactions.

Figure 32-151.Outbound Atomic Address Register (SRREG0_OATMADR)

Bit	Field Name	Reset State	Function
0:31	OAAD	0	Outbound Atomic Dword Address Bits 0:31 map to SRIO address 0:31. SRIO address 32:33 always = 0.

32.6.10.18 Outbound Atomic Destination ID Register (SRREG0_OATMDID)

This register contains the SRIO Destination ID for Outbound Atomic transactions.

Figure 32-152.Outbound Atomic Destination ID Register (SRREG0_OATMDID)

Bit	Field Name	Reset State	Function
0:23			Reserved
24:31	OADSTI	0	Outbound Atomic Destination ID

32.6.10.19 Outbound Atomic Data Register (SRREG0_OATMDOUT)

This register contains the data for Outbound Atomic Test and Swap transactions.

Figure 32-153.Outbound Atomic Data Register (SRREG0_OATMDOUT)

Bit	Field Name	Reset State	Function
0:31	OADO	0	Write data for Outbound Atomic Test & Swap transactions.

32.6.10.20 Local Atomic Control Register (SRREG0_LATMCTL)

This register is used to generate Local Atomic transactions.

Figure 32-154.Local Atomic Control Register (SRREG0_LATMCTL)

Bit	Field Name	Reset State	Function
0	LAEVEN	0	Local Atomic Event Enable Writing this bit causes the Bridge to perform one of 5 possible Atomic transactions to local memory. This bit is automatically cleared by the bridge when the local atomic command is complete, and a new command can be written.
1	LARDDN	0	Automatically set when Local Atomic Event Enable is set. This bit is automatically cleared by the bridge when data is available to be read from the SRREG0_LATMDIN register.
2:23			Reserved
24:27	LASIZ	0	Local Outbound Atomic Byte Enables 0000 Byte 0 Valid 0001 Byte 1 Valid 0010 Byte 2 Valid 0011 Byte 3 Valid 0100 Bytes 0-1 Valid 0110 Bytes 2-3 Valid 1000 Bytes 0-3 Valid **Note:** PLB bits 26:27 are equal to the SRIO 34-bit address bits 32:33.

Figure 32-154.Local Atomic Control Register (SRREG0_LATMCTL) (Continued)

28:31	LATRTY	0	Local Outbound Atomic Transaction Type 1000 ATMTS_R - Atomic Test and Swap with Response 1010 ATMINC_R - Atomic Increment with Response 1011 ATMDEC_R - Atomic Decrement with Response 1100 ATMSET_R - Atomic Set with Response 1111 ATMCLR_R - Atomic Clear with Response

32.6.10.21 Local Atomic Address Register (SRREG0_LATMADR)

This register contains the SRIO address for Local Atomic transactions.

Figure 32-155.Local Atomic Address Register (SRREG0_LATMADR)

Bit	Field Name	Reset State	Function
0:31	LAAD	0	Bits 0:31 of local SRIO 34-bit address Bits 32:33 are implied by the LASIZ field in the SRREG0_LATMCTL register bits 26:27. This address is translated to a PLB address like an incoming SRIO command's 34-bit address.

32.6.10.22 Local Atomic Data Out Register (SRREG0_LATMDOUT)

This register contains the data for Local Atomic Test and Swap transactions.

Figure 32-156.Local Atomic Data Out Register (SRREG0_LATMDOUT)

Bit	Field Name	Reset State	Function
0:31	LADO	0	Write Data out for Local Atomic Test and Swap transactions.

32.6.10.23 Local Atomic Data In Register (SRREG0_LATMDIN)

This register contains the read data for Local Atomic transactions.

Figure 32-157.Local Atomic Data In Register (SRREG0_LATMDIN)

Bit	Field Name	Reset State	Function
0:31	LADI	0	Read data for all Local Atomic transactions.

32.6.10.24 Inbound Transaction Timeout Register (SRREG0_INBTO)

This register contains the time out value for inbound SRIO transactions.

Figure 32-158.Inbound Transaction Timeout Register (SRREG0_INBTO)

Bit	Field Name	Reset State	Function
0:7			Reserved
8:31	INTRTV	0xFFFFFF	Number of PLB Clocks to determine the period of time out. Set to zero to avoid interrupts.

32.6.10.25 Response Completion Status Register (SRREG0_RSPSTAT)

This register provides status information for SRIO Outbound transactions.

Figure 32-159.Response Completion Status Register (SRREG0_RSPSTAT)

Bit	Field Name	Reset State	Function
0:24			Reserved
25:31	OFCCS	0	Outbound Response Completion Register 0b0000000 - Done OK w/o payload 0b0000x11 - Retry 0b0001x11 - Error 0b1cccccc - Done OK for Response with Payload where cccccc = number of beats to Write. (Should not see this value, as only responses without payload are captured.)

33. IIC Bus Interface

The PPC460EX/EXr/GT provides two inter-integrated circuit (IIC) bus interfaces, IIC0 and IIC1. These interfaces comply with the specifications contained in the Phillips ® Semiconductors document *The I^2C Bus and How to Use It* (including specifications—1995 update).

Each IIC bus has a two wire, bidirectional, open drain, low speed serial interface. The serial clock (IIC0SClk and IIC1SClk) and serial data (IIC0SDA and IIC1SDA) lines are bidirectional, to support multiple bus masters and to mix high- and low-speed devices on the same bus.

Each IIC interface (referred to as IIC to distinguish it from the Phillips I 2 C bus) supports the following standard and enhanced features:

- 100 kHz and 400 kHz operation
- 8-bit data transfers
- 7 bit and 10 bit addressing
- Slave transmitter and receiver
- Master transmitter and receiver
- Multiple bus masters

33.1 Overview of the I2C Bus

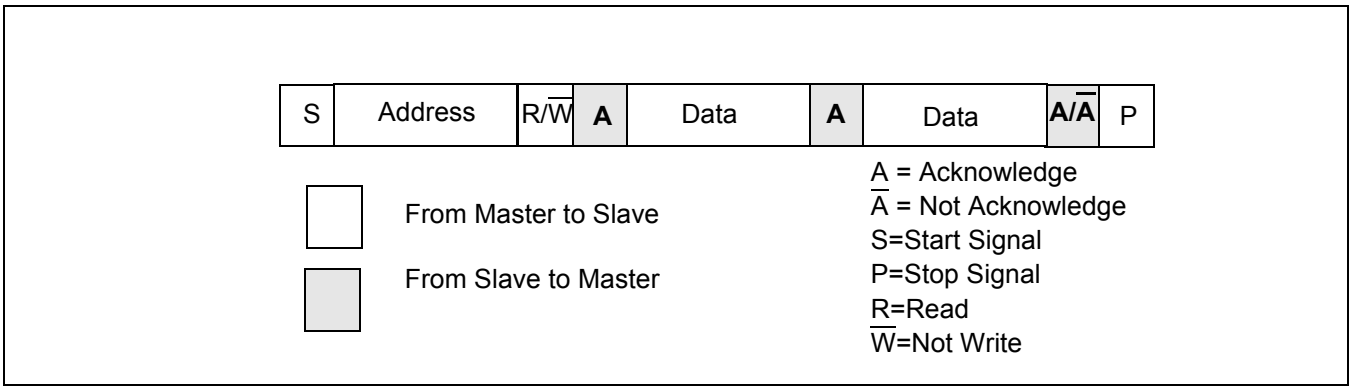
To better understand the functionality of the IIC controller, a high-level understanding of the I2C bus is needed. Following is a short summary of the information needed to understand how to use the IIC controller and the I2C bus. This summary is not, however, meant to replace the I2C bus specification.

As stated above, the I2C bus is a 2-wire interface that is composed of a clock line and a data line. The actions and function of this serial interface are such that it operates like a parallel multimaster bus interface. When the I2C bus is free, one or more masters start a transmission to a slave that is attached on the bus. During this transmission, the masters vie for ownership of the bus as they write data to or read data from a slave. One master wins ownership of the bus during this transmission. It is then free to do as many transfers as it desires before it must relinquish bus ownership. Once the master yields ownership of the

I2C bus, the bus is considered free. At this point, any other master, including the previous master, can attempt to initiate a new transfer and take control of the I2C bus.

A high-level diagram of a write operation on the I2C bus is shown in Figure 33-1.

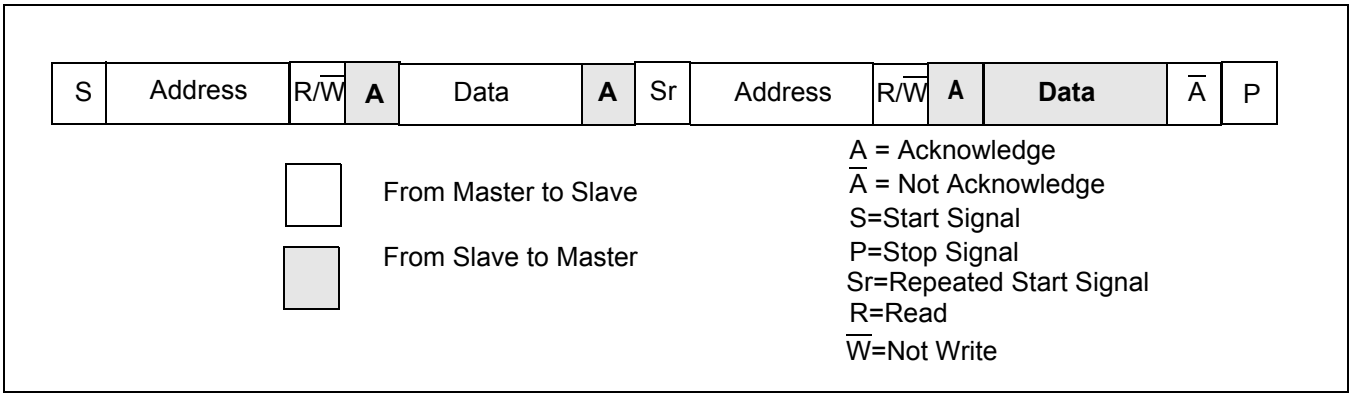
Figure 33-1. 7-bit Address Write Operation



The sequence of operation for all transfers must begin with a start signal. A start signal performs two functions on the bus. First, it tells other potential masters that the bus is busy and that they must wait to use the bus. Second, it alerts all slave devices that a new transfer is starting and that they must decode the next seven bits from the master to see if they are being addressed. The next seven bits are shown in the figure as the ADDRESS. The next, or eighth bit, indicates the type of transfer (R = Read = 1, W = Write = 0). In this case, the master sends a logic '0' to indicate it wants to write information to the slave. The ninth bit is driven by the slave that decodes its address. Note that if no slave was addressed, or if the addressed slave was busy, a not acknowledge (A) would be on the bus. In Figure 33-1, a slave sends an acknowledge (A) because it has decoded its address and it is able to accept data. The next item is eight bits, or one byte, of data followed by an acknowledge from the slave. In the figure, the master sends a second data byte, and the slave either sends an acknowledge or a not acknowledge. (A not acknowledge on the second byte, for example, would indicate that the slave cannot accept the second data byte. In this case the master would have to either end the transmission, resend the second data byte, or start a new transfer). When the master is done transferring data, it issues a stop signal to indicate that the I2C bus is free for use by another master. The addressed slave and all other masters see the stop signal and use it to determine whether the bus is free.

A high-level diagram of a write operation followed by a read operation on the I2C bus is shown in Figure 33-2.

Figure 33-2. 7-bit Address Write Operation Followed by a 7-bit Read Operation



As always, this sequence starts with a start signal. Here the master writes one byte to the slave. Instead of sending a stop signal after the write, as is the case in Figure 33-1, this master wants to perform a second operation on the I2C bus. The second operation begins when the master issues another start signal in place of the stop signal. The second start signal is called a repeated start because it follows a data transfer that was not ended by a stop signal. The repeated start is used so that the master ensures that it can retain ownership of the I2C bus. Note that this can be useful when a master must first write a number of bytes to a particular slave before it can read information from the slave. If the repeated start was not used, another master could obtain control of the bus and could reprogram the same slave thus interfering with the first master's operation. In the case in Figure 33-2, the master sends seven bits of address after the repeated start signal. The eighth bit, being the R/W, is a logic '1' to indicate that this is a read operation. A slave has decoded the address and is free to send the requested data, so it issues an acknowledge signal. Once the master reads the byte as driven by the slave, the master issues a not acknowledge to indicate that no more bytes are to be read at this time. Since the master is done, it then issues a stop signal to free the I2C bus for use by another master.

33.2 Master Operation

By setting a few registers, the IIC controller can initiate a transfer on the I2C bus for either a write or a read. The controller's logic handles all the I2C bus's protocol that is needed to perform the requested operation. This includes handling a simple 1-byte write operation or a multiple byte 10-bit read operation. Once the operation is concluded, the IIC controller sets status to indicate the results of the transfer and, if so enabled, issues an interrupt.

Before initiating a 1-byte write operation, or any operation, it is required that several registers be initialized according to the operation that is required in the system. Among the registers that need to be set are the clock divide, the interrupt enable and the mode control. Note that if this IIC controller is going to be addressed as a slave on the I2C bus then, prior to setting the mode control register, the low and high slave address registers should be set with the address value that the core should decode and reply to when it detects operations on the I2C bus. While the initialization of these registers is normally only done once after a reset, they can be reprogrammed at any time.

The following steps are required when initializing the IICx controller after reset.

1. Set IICx Low Master Address and IICx High Master Address.
2. Set IICx Low Slave Address and IICx High Slave Address.
3. Clear any active interrupts in the IICx Status register (write IICx_STS=0x0A).
4. Clear all writeable status bits in the IICx Extended Status register (write IICx_EXTSTS=0x8F).
5. Set the IICx Clock Divisor register (IICx_CLKDIV).
6. Clear all interrupt mask bits in the IICx Interrupt Mask register (write IICx_INTRMSK=0x00).
7. Clear the IICx Transfer Count register (write IICx_XFRCNT=0x00).
8. Clear status set in the IICx Extended Control and Slave Status register (write IICx_XTCNTLSS=0xF0).
9. Initialize the IICx Mode Control register.
10. Clear the IICx Control Register (write IICx_CNTL=0x00).

Note: It is imperative that after a reset operation the IIC controller exits the unknown I2C bus state. This occurs when the controller detects activity on the I2C bus. In a multiple master system, it is probably best to let the controller detect and react to activity on the I2C bus. In single master systems, however, (or in other special cases that are only known to the systems designer) the IIC controller never detects any such activity, it stays in the unknown state forever, and thus the controller never responds to any attempts to start a transfer on the I2C bus. To enable transfers in such cases, the software must force the IIC controller out of this unknown state by setting the exit unknown I2C bus state bit, bit 1 in the mode control register.

Once this setup is done, a data transfer operation can be requested. If, for example, a 1-byte write with 7-bit addressing needs to be performed, then the lo master address register should be programmed with the address value that the master logic sends out onto the I2C bus. Note that if 10-bit addressing is going to be used, then the hi master address register should also be programmed. Because the values in these registers are never destroyed after an operation, it is possible to reuse the contents of these address registers. Thus if successive operations to the same address are performed, then the software only needs to set these registers once. Each transfer that is requested of the IIC controller then reuses the values in these registers when it commences an operation on the I2C bus. Next, the master data buffer must be loaded with the byte of data to be written out onto the I2C bus. Finally, the control register should be written with the pending transfer bit set to a logic '1' to actually initiate the transfer. At this point the IIC controller sends out the address onto the I2C bus using the value in the lo master address register. Just before the eighth address bit is transmitted, the IIC controller substitutes the read/not write (R/nW) bit. The IIC controller automatically handles sending out the start signal just before the address byte, detecting loss of arbitration for the I2C bus during the transfer, and checking the Slave acknowledgment of the address. If arbitration was not lost and if a positive address acknowledge was received, then the IIC controller transmits the one byte of data that was written into the master data buffer. The IIC controller automatically handles

checking for the slave's acknowledgment of the data byte as well as issuing the stop signal after the acknowledge signal. Once the stop signal has been issued, the IIC controller updates its status registers and, if so enabled, activates its interrupt signal.

Performing the more complex 10-bit read operation on the I2C bus is just as simple for the processor as a write with 7-bit addressing. The master data buffer should be emptied of all data before the transfer is initiated by the processor (but this is not required). If a different slave needs to be addressed then the lo and hi master address registers need to be reprogrammed. Finally, the control register must be written with the 10/7 bit addressing bit set to logic '1', the count bits set accordingly, the R/nW bit set to logic '1', and the pending transfer bit set to a logic '1' to actually initiate the transfer. At this point, the IIC controller starts the transfer on the I2C bus. The controller's logic handles the complex protocol that is used to initiate and perform a 10-bit address read operation. This includes issuing the start signal, sending the high-order address byte with the transfer set to a write (as per I2C bus protocol, all 10-bit address reads must be proceeded by a 10-bit address write to select the slave), sending the low-order address byte, detecting acknowledge on the transfer of each address byte, sending a second start signal (also called a repeated start), resending the high-order address byte with the transfer set to a read, reading the requested number of bytes, sending a not-acknowledge when the last byte is read, and then issuing the stop signal. Once the stop signal has been issued, the IIC controller updates its status registers and, if so enabled, activates its interrupt signal.

The processor can leave old information in the extended status and transfer count registers when a new master operation is requested. The completion of a transfer by the IIC controller's master logic causes these registers to be updated automatically. Any old information is overwritten when the operation is ended. This is not the case for the status register. It must be cleared prior to beginning of a new transfer.

To start a master transfer via the IIC controller, the pending transfer bit must be set to a logic '1' in the control register. The instant this bit is set, the master logic initiates a transfer on the I2C bus, and the core's logic works as determined by the contents of the registers and buffers. For example, if a write operation is requested and there is garbage in the master data buffer then the IIC controller writes garbage out onto the I2C bus. Thus it is imperative that the control register is the last item programmed by the processor's software. Note, if in the control register the pending transfer bit is set along with both the hold bus ownership bit and the chained bit then the hold bus ownership bit takes precedence over the chained bit.

Setting the hold bus ownership bit (IICx_CNTL[BCL]) when a transfer is requested informs the IIC controller master logic how to end this transfer as opposed to how to start it. In effect, setting the hold bus ownership bit instructs the IIC controller master logic to not send the stop signal at the conclusion of this transfer. Note that if the stop signal is not sent, then the current master retains ownership of the I2C bus. The hold bus ownership bit can be very useful in multiple master systems. Many slaves require multiple data transfers before an operation to them can be performed. Thus its use is required to gain and hold ownership of the I2C bus such that a slave can be programmed without interruption. These multiple data transfers are performed via use of the I2C bus's repeated start signal. The actual repeated start signal is issued on the I2C bus when the software sets the pending transfer bit for the second time. Also, if this second operation has the hold bus ownership bit set to a logic '1' then the stop signal is not set at the end of this second transfer. Note that once an operation is requested with the hold bus ownership bit set, a non-hold bus ownership transfer should be done so that the IIC controller can issue a stop signal and thus free the I2C bus for use by other masters. In the current implementation, if the software performs a transfer with the hold bus ownership bit set to a '1', then the software must follow this transfer with either a non-hold bus ownership operation or a halt operation. Either of these operations causes the IIC controller to issue a stop signal on the I2C bus (and thus free it for use by other masters), to set status and an interrupt if so enabled.

The software should set the chained bit to a logic '1' when it wants the IIC controller master logic to perform a transfer that is longer than four bytes. There are certain restrictions on what operations can follow a chained transfer. It is normal, for example, for a chained write of four bytes to be followed by another chained or non-chained write of some number of bytes.

The intended use of the chained bit is to allow for data transfers that are longer than four bytes. Once the initial chained read or write has ended, the IIC controller enters the paused state. The paused state is a subset of the master transfer state, so when this state is entered, the IIC controller is in the middle of a transfer. Thus, it owns the bus and is holding ownership of the I2C bus frozen by forcing the SCL signal low. At this point, the IIC controller needs to be restarted with another transfer. At this point, it is also possible to issue a halt to the IIC controller. If the IIC controller is doing a chained write, the operation is ended on the I2C bus via a Stop signal. The IIC controller gives up ownership of the bus. If the core was doing a chained read operation, issuing a halt is NOT RECOMMENDED. This is because doing a halt could cause the IIC controller to get hang while trying to issue a stop. Whether or not a hang occurs depends upon how the slave from which the user is reading is driving the I2C bus. It could be driving the SDA line low with the next bit of the next byte to be read. This is permitted because in a read the slave drives the bus until the master signals a non-acknowledge. Note that in a chained read, the last byte that ends the chained read is acknowledged and thus the slave is still driving the bus, thereby preventing the master from generating the stop signal. If the next bit was a logic '1' then the SDA line is high and the master is able to issue a stop signal and no hang-up condition occurs.

Table 33-1. Permitted Combinations of Operations

Operation N	Operation N+1	Permitted?
Write	Write	Yes
Write	Chained Write	Yes
Write	Hold Bus Ownership Write	Yes
Write	Read	Yes
Write	Chained Read	Yes
Write	Hold Bus Ownership Read	Yes
Chained Write	Write	Yes
Chained Write	Chained Write	Yes
Chained Write	Hold Bus Ownership Write	Yes
Chained Write	Read	No
Chained Write	Chained Read	No
Chained Write	Hold Bus Ownership Read	No
Hold Bus Ownership Write	Write	Yes
Hold Bus Ownership Write	Chained Write	Yes
Hold Bus Ownership Write	Hold Bus Ownership Write	Yes
Hold Bus Ownership Write	Read	Yes
Hold Bus Ownership Write	Chained Read	Yes
Hold Bus Ownership Write	Hold Bus Ownership Read	Yes
Read	Write	Yes
Read	Chained Write	Yes
Read	Hold Bus Ownership Write	Yes
Read	Read	Yes

Table 33-1. Permitted Combinations of Operations (Continued)

Operation N	Operation N+1	Permitted?
Read	Chained Read	Yes
Read	Hold Bus Ownership Read	Yes
Chained Read	Write	No
Chained Read	Chained Write	No
Chained Read	Hold Bus Ownership Write	No
Chained Read	Read	Yes
Chained Read	Chained Read	Yes
Chained Read	Hold Bus Ownership Read	Yes
Hold Bus Ownership Read	Write	Yes
Hold Bus Ownership Read	Chained Write	Yes
Hold Bus Ownership Read	Hold Bus Ownership Write	Yes
Hold Bus Ownership Read	Read	Yes
Hold Bus Ownership Read	Chained Read	Yes
Hold Bus Ownership Read	Hold Bus Ownership Read	Yes

Steps for performing write, repeated start, read:

1. Clear the Stop Complete bit (write IICx_STS[SCMP]=1).
2. Poll the Pending Transfer bit (IICx_STS[PT]) until it is 0, indicating that there are no pending transfers.
3. Set the Flush Master Data Buffer bit (write IICx_MDCNTL[FMDB]=1) to clear the buffer.
4. Place the address of the IIC slave to be accessed in the IICx Low Master Address (IICx_LMADR) and IICx High Master Address (IICx_HMADR) registers. The IICx_HMADR and IICx_LMADR[A7] are only used for 10-bit addressing.
5. Initialize the IICx Control Register (IICx_CNTL) to start the write transfer. The IICx_CNTL contains several configuration options controlling the write transfer:

   Select a transfer as the bus operation
   Clear the Halt Master Transfer bit (write IICx_CNTL[HMT]=0) to perform a transfer.

   Select the address mode
   Two address modes are supported, 7-bit addressing (IICx_CNTL[AMD]=0) or 10-bit addressing (IICx_CNTL[AMD]=1).

   Set the transfer count
   One to four bytes can be written per a transfer. The Transfer Count bit field specifies the number of bytes (IICx_CNTL[TCT])

   Select how the transfer ends
   Set (IICx_CNTL[BCL] = 1) to prevent the IIC controller from generating a stop. When IICx_CNTL[BCL] = 1, the IIC controller maintains ownership of the bus and the next transfer begins with a repeated start.

Specify if this transfer is the last transfer in a sequence of transfers
For the last write in a chain or a single transfer, clear the Chain Transfer bit (write IICx_CNTL[CHT]=0). Clearing this bit instructs the IIC controller to place a stop condition after the transfer. For a write chained to a subsequent write, set the Chain Transfer bit (write IICx_CNTL[CHT]=1). No stop condition is placed between chained transfers.

Enable a write transfer
Clear the Read/Write bit (write IICx_CNTL[RW]=0) for a write.

Begin the transfer
Set the Pending Transfer bit (write IICx_CNTL[PT]=1). Once enabled, the write transfer begins as soon as the IIC bus is free.

6. Poll the Pending Transfer bit (IICx_STS[PT]) until it is 0, indicating that the read transfer completed.
7. Initialize the IICx Control Register (IICx_CNTL) to start the read transfer. The IICx_CNTL contains several configuration options controlling the read transfer:

   Select a transfer as the bus operation
   Clear the Halt Master Transfer bit (write IICx_CNTL[HMT]=0) to perform a transfer.

   Select the address mode
   Two address modes are supported, 7-bit addressing (IICx_CNTL[AMD]=0) or 10-bit addressing (IICx_CNTL[AMD]=1).

   Set the transfer count
   One to four bytes can be read per a transfer. The Transfer Count bit field specifies the number of bytes (write IICx_CNTL[TCT]).

   Select how the transfer ends
   Set (IICx_CNTL[BCL] = 0) to force the IIC controller to generate a stop at the end of the transfer.

   Specify if this transfer is the last transfer in a sequence of transfers
   For the last read in a chain or a single transfer, clear the Chain Transfer bit (write IICx_CNTL[CHT]=0). Clearing this bit instructs the IIC controller to place a stop condition after the transfer. For a read chained to a subsequent read, set the Chain Transfer bit (write IICx_CNTL[CHT]=1). No stop condition is placed between chained transfers.

   Enable a read transfer
   Set the Read/Write bit (write IICx_CNTL[RW]=1) for a read.

   Begin the transfer
   Set the Pending Transfer bit (write IICx_CNTL[PT]=1). Once enabled, the read transfer begins as soon as the IIC bus is free.

8. Poll the Pending Transfer bit (IICx_STS[PT]) until it is 0, indicating that the read transfer completed.
9. Read the Error status bit (IICx_STS[ERR]) to ensure the transfer completed error free.
   - If an error is indicated (IICx_STS[ERR]=1), read the Lost Arbitration, Incomplete Transfer and Transfer Aborted bits (IICx_EXTST[LA,ICT,XFRA]) to determine how to recover from the error.
   - If no error is indicated (IICx_STS[ERR]=0), read the data from the Master Data Buffer (IICx_MDBUF). The IICx Transfer Count Register (IICx_XFRCNT[MTC]) indicates the number of bytes available in the Master Data Buffer. The IICx_MDBUF can be read with byte (lbz) or half word (lhz) operations. Full word operations are not supported.

Note: The IIC controller does not implement a bus time-out function. A software timer should be implemented to enforce a timeout for steps that poll the Pending Transfer status bit (IICx_STS[PT]).

Steps for performing an IIC write to an IIC slave:

1. Clear the Stop Complete bit (write IICx_STS[SCMP]=1).
2. Poll the Pending Transfer bit (IICx_STS[PT]) until it is 0, indicating that there are no pending transfers.
3. Set the Flush Master Data Buffer bit (write IICx_MDCNTL[FMDB]=1) to clear the buffer.
4. Write 1 to 4 bytes of data into the Master Data Buffer (IICx_MDBUF). The IICx_MDBUF can be written with byte (stb) or half word (sth) operations. Full word operations are not supported.
5. Place the address of the IIC slave to be accessed in the IICx Low Master Address (IICx_LMADR) and IICx High Master Address (IICx_HMADR) registers. The IICx_HMADR and IICx_LMADR[A7] are only used for 10-bit addressing.
6. Initialize the IICx Control Register (IICx_CNTL) to start the write transfer. The IICx_CNTL contains several configuration options controlling the write transfer:

   Select a transfer as the bus operation
   Clear the Halt Master Transfer bit (write IICx_CNTL[HMT]=0) to perform a transfer.

   Select the address mode
   Two address modes are supported, 7-bit addressing (IICx_CNTL[AMD]=0) or 10-bit addressing (IICx_CNTL[AMD]=1).

   Set the transfer count
   One to four bytes can be written per a transfer. The Transfer Count bit field specifies the number of bytes (IICx_CNTL[TCT])

   Select how the transfer ends
   Set (IICx_CNTL[BCL] = 0) to force the IIC controller to generate a stop at the end of the transfer. Or set (IICx_CNTLL[BCL] = 1) to prevent the IIC controller from generating a stop. When IICx_CNTLL[BCL] = 1, the IIC controller maintains ownership of the bus and the next transfer begins with a repeated start.

   Specify if this transfer is the last transfer in a sequence of transfers
   For the last write in a chain or a single transfer, clear the Chain Transfer bit (write IICx_CNTL[CHT]=0). Clearing this bit instructs the IIC controller to place a stop condition after the transfer. For a write chained to a subsequent write, set the Chain Transfer bit (write IICx_CNTL[CHT]=1). No stop condition is placed between chained transfers.

   Enable a write transfer
   Clear the Read/Write bit (write IICx_CNTL[RW]=0) for a write.

   Begin the transfer
   Set the Pending Transfer bit (write IICx_CNTL[PT]=1). Once enabled, the write transfer begins as soon as the IIC bus is free.
7. Poll the Pending Transfer bit (IICx_STS[PT]) until it is 0, indicating that the write transfer completed.
8. Read the Error status bit (IICx_STS[ERR]) to ensure the transfer completed error free.
   - If an error is indicated (IICx_STS[ERR]=1), read the Lost Arbitration, Incomplete Transfer and Transfer Aborted bits (IICx_EXTST[LA,ICT,XFRA]) to determine how to recover from the error.
   - If no error is indicated (IICx_STS[ERR]=0), read the IIC Transfer Count Register (IICx_XFRCNT[MTC]). The IICx_XFRCNT[MTC] bit field indicates the number of bytes written to the slave device.

Note: The IIC controller does not implement a bus time-out function. A software timer should be implemented to enforce a timeout for steps that poll the Pending Transfer status bit (IICx_STS[PT]).

Steps for performing an IIC read from an IIC slave:

1. Clear the Stop Complete bit (write IICx_STS[SCMP]=1).
2. Poll the Pending Transfer bit (IICx_STS[PT]) until it is 0, indicating that there are no pending transfers.
3. Set the Flush Master Data Buffer bit (write IICx_MDCNTL[FMDB]=1) to clear the buffer.
4. Place the address of the IIC slave to be accessed in the IICx Low Master Address (IICx_LMADR) and IICx High Master Address (IICx_HMADR) registers. The IICx_HMADR and IICx_LMADR[A7] are only used for 10-bit addressing.
5. Initialize the IICx Control Register (IICx_CNTL) to start the read transfer. The IICx_CNTL contains several configuration options controlling the write transfer:

   Select a transfer as the bus operation
   Clear the Halt Master Transfer bit (write IICx_CNTL[HMT]=0) to perform a transfer.

   Select the address mode
   Two address modes are supported, 7-bit addressing (IICx_CNTL[AMD]=0) or 10-bit addressing (IICx_CNTL[AMD]=1).

   Set the transfer count
   One to four bytes can be written per a transfer. The Transfer Count bit field specifies the number of bytes (IICx_CNTL[TCT])

   Select how the transfer ends
   Set (IICx_CNTL[BCL] = 0) to force the IIC controller to generate a stop at the end of the transfer. Or set (IICx_CNTLL[BCL] = 1) to prevent the IIC controller from generating a stop. When IICx_CNTLL[BCL] = 1, the IIC controller maintains ownership of the bus and the next transfer begins with a repeated start.

   Specify if this transfer is the last transfer in a sequence of transfers
   For the last write in a chain or a single transfer, clear the Chain Transfer bit (write IICx_CNTL[CHT]=0). Clearing this bit instructs the IIC controller to place a stop condition after the transfer. For a write chained to a subsequent write, set the Chain Transfer bit (write IICx_CNTL[CHT]=1). No stop condition is placed between chained transfers.

   Enable a read transfer
   Clear the Read/Write bit (write IICx_CNTL[RW]=1) for a read.

   Begin the transfer
   Set the Pending Transfer bit (write IICx_CNTL[PT]=1). Once enabled, the read transfer begins as soon as the IIC bus is free.
6. Poll the Pending Transfer bit (IICx_STS[PT]) until it is 0, indicating that the write transfer completed.
7. Read the Error status bit (IICx_STS[ERR]) to ensure the transfer completed error free.
   - If an error is indicated (IICx_STS[ERR]=1), read the Lost Arbitration, Incomplete Transfer and Transfer Aborted bits (IICx_EXTST[LA,ICT,XFRA]) to determine how to recover from the error.
   - If no error is indicated (IICx_STS[ERR]=0), read the IIC Transfer Count Register (IICx_XFRCNT[MTC]). The IICx_XFRCNT[MTC] bit field indicates the number of bytes available in the Master Data Buffer. The IIC0_MDBUF can be read with byte (lbz) or half word (lhz) operations. Full word operations are not supported.

Note: The IIC controller does not implement a bus time-out function. A software timer should be implemented to enforce a timeout for steps that poll the Pending Transfer status bit (IICx_STS[PT]).

33.3 Multimaster Operations

Multimaster operations occur in systems that have more than one master on the I2C bus. Due to the nature of how the I2C bus functions these systems can be an order of magnitude harder to manage and program. This section explains some of the complexities of this area of the I2C bus. Three areas of complexity in multimaster systems are bus arbitration, initialization after a reset, and error conditions. It is important to understand these issues prior to a discussion of how the IIC controller reacts to these conditions and how to avoid problems in the device driver software.

33.3.1 Bus Arbitration

Arbitration on the I2C bus is different from many other common buses. In these common buses masters compete for ownership of the bus during an arbitration cycle. This cycle can be prior to a new data transfer or in parallel with a current master's last transfer. In either case one particular master is given definitive ownership of the bus. This is not the case in the I2C bus. On the I2C bus arbitration is done during every SCL cycle in which a master is driving the SDA signal. A master detects a loss of arbitration when it sees a low on the line when it was trying to drive a high. The net effect of this is that a master only knows that it has lost arbitration, but is never told that it has won bus arbitration. Thus, a master assumes it has won ownership of the bus if it never sees that it has lost. Furthermore, since the arbitration occurs during all address bits (the R/nW bit, all data bits written by a master in a write, and any acknowledge signals from a master in a read operation) and operations can be of any length, this arbitration process can be very time consuming.

Another complicating factor is that the I2C bus contains very few addresses, and as such a slave uses only one address on the bus. In addition, many slaves need multiple numbers of bytes to know what is requested of them. The single address of the slave means that the data written to a slave is used either as some type of sub addressing mechanism, or is used as some type of command sequence, or both. These mechanisms are undefined by the I2C bus and are the sole responsibility of the slave.

The combination of bus arbitration and slave operation form the central issue in the complexity of multimaster I2C bus systems. Of course, the complexity is also determined by the types, numbers, and usage of slaves in the system. For example, if each master addresses a set of mutually exclusive slaves then the system complexity is vastly reduced. A very complex problem can be caused if the masters share many slaves, if these slaves need multibyte write operations to set them up, and if these multibyte sequences are common (and varied). The following example illustrates this complexity:

> Consider a system that has two I2C masters, called M1 and M2, and one I2C slave, called S1. We assume that to read from S1 you need to write at least three bytes to S1. Assume that both M1 and M2 start at exactly the same time and that they both write the same three bytes to S1. Assume that after byte 3 is sent that neither master issues a stop. Neither master issues a stop because they both want to own the I2C bus. Next assume that both M1 and M2 issue repeated starts at exactly the same time. In this new transfer, M1 is doing a read from S1, and M2 is doing a write. In this case M1 loses arbitration and M2 wins (M1 sees a low during the R/nW bit and thus detects loss of bus arbitration). M2 now owns the bus and M1 exists to some non-transfer state. M1 must wait until M2 issues a stop and then retry the failed read. To redo the read M1 has to back up and first redo the 3-byte write operation that sets up S1. This needs to be done because while M2 owned the bus it could have sent many bytes of data to S1. Then once M1 tries to do its read S1 is in an unknown state (M1's initial three bytes of set-up information have potentially been over-written). Thus in this case, and for this system, M1's device driver code has to keep track of the 3-byte string until the read is successfully completed.

The above example, with only three bytes to one slave, is rather simple, but in general the data strings could be very long and could involve multiple write, read, write, read operations. What needs to be kept track of is a function of the system. Unless the system is able to maintain data pointers and other structures, it could be very hard to design them in after most of the system code has been put into place.

33.3.2 Initialization After a Reset

In a single master system when the IIC controller is reset you can simply program the IIC controller's registers and advance its main state machine from the unknown I2C bus state to the free I2C bus state. This is not the case in a multimaster system. Since an I2C bus system can be distributed in physical space and potentially be spread across multiple machines, the I2C master might have been the only one to have been reset, or powered on. Thus you cannot generally assume that the bus is free after a reset. This is a function of the system and its architecture1. The state of the bus after a reset can be automatically determined by the IIC controller. Once its clock divide register is programmed the core starts looking for activity on the I2C bus. Table 33-2 summarizes the actions and reactions of the IIC controller after a reset.

Table 33-2. IIC Controller Actions and Reactions Following Reset

Clock Divide Register Programmed?	Action	Reaction
No	Don't Care	IIC controller remains in the unknown I2C bus state.
Yes	Exit Unknown I2C Bus State Bit is set to a logic '1'	IIC controller advances to the free I2C bus state.
Yes	IIC controller detects high to low transition on the SCL signals	IIC controller advances to the busy I2C bus state.
Yes	IIC controller detects Start	IIC controller advances to the busy I2C bus state.
Yes	IIC controller detects Stop	IIC controller advances to the free I2C bus state.

Initially it may seem very simple how to handle things after a reset, but it is not as easy as just letting the IIC controller do it for you. Due to the slow data transfer rates used on the I2C bus and the potential latencies in unready masters or slaves, quite a long period of inactivity could occur. This long latency combined with potential failures on the I2C bus causes a problem because it can be difficult to determine at what point the bus has become simply inactive, versus in use, versus hung.

An inactive bus is one in which there is no master. The bus is actually in the free I2C bus state but the IIC controller cannot detect this because of the state of the SCL and SDA lines and the resulting intersection of this state with another I2C bus state. In this state the SCL and SDA lines remains high.

An in use bus is one in which there is a current bus master. The bus is actually in the busy I2C bus state but the IIC controller cannot detect this because of the state of the SCL and SDA lines and the resulting intersection of this state with another I2C bus state. In this state, the lines can remain constant (either high, either low, one high the other low) or they may change state. If the lines change state then the IIC controller reacts as shown in Table 33-2. If the lines do not change state then the core does not know how to react. Note that the values of the states of the SCL and SDA signals can look identical to the inactive bus state.

A hung bus is one in which there may be none, one, or more current bus masters. The bus is actually in the busy I2C bus state but the IIC controller cannot detect this because of the state of the SCL and SDA lines, and the resulting intersection of this state with the other I2C bus states. In this state the lines remain constant (either high, either low, one high the other low). Thus this state can look like either the inactive or in use state. The device driver code in a master must be designed with some type of system-dependent time-out function. That is, after a reset is done, if the IIC controller remains in the unknown I2C bus state after the time-out is hit then the code must perform some type of error recovery function.

33.3.3 Error Conditions

Multimaster systems suffer from some errors that cannot happen in single master systems. Since these systems also incur problems seen in single master systems, they are by nature more complex. These errors may include: time-outs (due to a hang either after reset or in a data transfer), illegal start/stop signals, non acknowledgment of an address, incomplete transfers and loss of bus arbitration.

- Time-outs: A time-out can occur due to either being stuck in the unknown I2C bus state after a reset or being stuck in a data transfer. While the first error happens only in a multimaster system, the second error, a hang in a data transfer, can occur in either a single or a multimaster system. One reason why you could get stuck in a data transfer is that a slave or master device is holding the SCL signal low and is not releasing it to high[1]. Another reason is that a device is holding the SDA signal low and is preventing the stop or start signal from being issued[2]. Time-outs can also occur when a requested master transfer starts, loses arbitration, and never gets reattempted due to the current master never issuing stop on the I2C bus[3].
- Illegal start/stop signals: Illegal signals are the result of either noise on the SCL and SDA lines, malfunctions of device hardware, or mistakes by software that is controlling masters in the system. The single characteristic of these signals is that they happen at a point in the protocol where they are not permitted to happen[4]. When these illegal signals happen, the response of an I2C bus device is not defined by the I2C bus specification, so in accordance with the I2C specification the response is left to the designer of each device.
- Non acknowledgment of an address: Every time a byte of address information is sent out on the I2C bus, all slaves decode the address to see if they are being accessed. A non acknowledgment occurs when no slave responds with an acknowledgment of the address byte. This can be caused by having an address which does not exist in the system, a slave that is currently too busy to accept or provide data[5], or a malfunction in the addressed slave.
- Incomplete data transfers: An incomplete data transfer happens when a master is not able to write all of the data that it was programmed to send. It can also occur in a read when a master is not able to obtain all of the data which it was programmed to receive.
- Loss of bus arbitration: Loss of bus arbitration occurs when a master is attempting to drive the SDA signal high and it sees that the SDA signal is actually at a low state. When this happens the master that detects it is said to have lost arbitration. If the losing master has a slave device then the slave logic must be ready to respond to the requested transfer. The winning master, or masters, continue with their desired transfer. Arbitration is performed whenever a master is driving the SDA signal. This includes all address bits, the R/nW bit, all data bits in a write, and all acknowledgment bits in a read. Arbitration on the I2C bus is thus constantly being done throughout an operation. Masters are never told that they own the bus; rather, they are told that they lost. A master thus wins ownership of the bus by never being told it lost[6].

Note 1: Devices are permitted to hold SCL low in order to pause or to slow down the transfer. If a device needs to hold SCL low to pause because it is busy, there is no information in the I2C specification covering how long SCL may be held low. If the SCL signal is left low a long time, you need to determine if it is busy or it has failed. The answer is a function of how the devices in the system operate.

Note 2: During a read operation, a slave drives the SDA signal to send data to the master. The slave is required to release the SDA signal when the master signals non-acknowledge on the last data byte to be read. It is possible that the slave cannot release SDA due to a malfunction.

Note 3: Some errors occur such that all masters think the bus is busy. Thus no master is in control and all are waiting for each other to free up the I2C bus. For these cases the system needs one or more masters who are looking for global failures on the I2C bus. Such masters can be thought of as "super masters" since they react to and try to recover from the actions and inactions of all masters in the system. A super master is, in effect, the master of the masters.

Note 4: A highly likely case is one in which one master, called Mr, is reset when another master, called Mt, is doing a data transfer. It is then possible that the Mr master is programmed to do a transfer

at a point when the SDA and SCL signals have been high for quite some time. In this case Mr thinks that the bus is free and starts its transfer. The net result is that the Mt master and the current slave see a Start in the middle of a transfer. The start issued by Mr is illegal as seen by the slave and Mt.

Note 5: Slaves are permitted by the I2C specification to non-acknowledge their address if they are busy. For these slaves, this might be a normal condition, not an error.

Note 6: The more data a master transfers without loss of arbitration is not an ensurance of owning the bus. Successful transmission of successive bits of address or data simply raise the probability that the master has won the bus. This probability asymptotically approaches, but never reaches, 100%.

33.3.4 Reactions to Loss of Arbitration, Reset and Error Conditions

When the IIC controller detects a loss of arbitration it immediately clears the pending transfer bit and sets the lost arbitration bit in the extended status register. An interrupt is also set if bit 1 in the interrupt enable register is set to a logic '1'. The status register contains: a cleared pending transfer bit, a set IRQ active bit (if this type of interrupt was enabled), a set error bit, a cleared, halted, or stopped bit (no stop was issued by the master so this bit is not set); and a master data buffer full or master data buffer has data set to reflect how much data is currently inside the master data buffer.

Loss of I2C bus arbitration causes the immediate termination of the requested master's transfer. If arbitration was lost during an address transfer then any data that was intended to be written is still inside the master data buffer. If arbitration was lost during the sending of data then the master data buffer still contains those bytes that were not totally sent. This residual data should be flushed from the buffer as the entire transfer has to be redone. If arbitration was lost during the acknowledge on the last byte of data in a read operation then the master data buffer contains the data that was read.

Loss of I2C bus arbitration during an address byte also causes the master to switch into slave mode. This happens independent of the setting of the enable slave mode bit in the mode control register. If the slave was not enabled then the IIC controller e simply does not acknowledge the address. If the slave was enabled then the address is decoded and the slave logic responds accordingly.

When the core is reset, either by the reset input to the core or by setting the soft reset bit in the extended control and slave status register to a logic '1', the logic and most registers inside the core are initialized. The state machines enter the unknown I2C bus state and affected registers have their bits set to logic '0' or '1'. Note that the master and slave data buffers are set to the empty state but they are not initialized to contain any particular values. The buffers should be thought of as containing junk or uninitialized data. After the reset is completed, the software has to initialize the registers inside the IIC controller. This includes determining how and when to advance the state machines out of the unknown I2C bus state.

The IIC controller uses a digital sampling technique to perform the 50 ns glitch filtering as required in v1.0 of the I2C Bus Specification. In addition a 3 out of 5 majority voting technique is used to ensure that information is sampled correctly on the I2C bus. Even with these techniques it is still possible for the IIC controller to detect invalid changes of state on the SCL and SDA signals. As stated previously, these invalid or illegal signals have various causes. Whatever the cause, illegal signals manifest themselves such that the IIC controller can see them as either illegal start or stop signals. The IIC controller detects these, immediately aborts its operation, and enters the busy I2C bus state. This termination occurs if the core is functioning as a master or as a slave. In addition to entering the busy I2C bus state, the appropriate illegal start or stop bit is set in the extended status register. The appropriate data buffer is automatically flushed, or emptied, of any residual data. The appropriate transfer count is cleared to '0'. An illegal signal during a slave transfer sets the appropriate slave complete status bit in the extended control and slave status register. In addition, an interrupt is generated if this type of interrupt is enabled in the interrupt enable register. The main issue associated with illegal signals is to prevent false data from either being sent or received. Allowing false data through to the processor causes a failure in the system that occurs far after

the actual failure on the I2C bus, and would therefore be extremely hard to debug, detect and recover from. A cleaner, and effectively faster, approach is to stop the core's operation and force the immediate retry of the transfer.

Entering the busy I2C bus state causes a transfer to be retried regardless of who was the master on the I2C bus at the time of the failure. If this IIC controller was a slave at the time of the failure then the transfer is non acknowledged thus forcing the master to sense a problem and thus redo the transfer. If the controller was a master at the time of the failure then the termination of the operation and resulting error status permit the processor to detect the error and thus discard the current and any previously received data.

Before the IIC controller's master logic can perform a data transfer, it must send an address on the I2C bus and get an acknowledgment of the address. Two potential reasons that may cause the master to see a non acknowledgment of the address are that no slave may exist at that address or, more likely, that the slave is too busy to send or receive data at this time. When a non acknowledgment of the address is detected the master immediately terminates the requested transfer. The address non-acknowledged bit is set in the extended status register. In the status register the pending transfer bit is cleared, the error bit is set, the halted or stopped bit is clear, and the master data buffer bits are set to reflect the amount of data that is in the buffer. In addition, if bit 1 in the interrupt enable register was set to a logic '1' then the IRQ active bit is set in the status register to reflect the fact that an interrupt was generated.

When the software instructs the IIC controller's master to perform a data transfer, it must set the count bits in the control register. When the master's transfer is ended, the logic compares the actual number of bytes transferred, as stored in the transfer count register, with the desired number of bytes in the count bits. If the transfer ended before the desired number of bytes were transferred then the incomplete transfer bit is set in the extended status register. In the status register, the pending transfer bit is cleared, the error bit is set, the halted or stopped bit is set if a stopped signal was sent, and the master data buffer bits are set to reflect the amount of data that is in the buffer. In addition, if bit 1 in the interrupt enable register was set to a logic '1', then the IRQ active bit is set in the status register to reflect the fact that an interrupt was generated. The transfer count register can be read to determine how many bytes were sent or received.

33.3.5 Methods to Simplify Complexity in Multimaster Device Driver Code

One major complexity in managing a multimaster interface is handling the need to redo a transfer. Associated with this need is the fact that a master can lose ownership of the I2C bus after a potentially large number of bytes. If the size of the transfer could be limited to some fixed, small number then the management of this interface would be greatly simplified.

The simplest way to limit the amount of information that may have to be resent is to have each master transmit a unique sequence of information. If this unique string precedes any access to a shared device then it ensures that the I2C bus arbitration process yields a winner of the bus early in the data transmission. One good side effect of this is to ensure that the data that needs to be retried is of a short, known length. Another benefit is to speed throughput on the bus; because the repeated transfer is shorter, more information as a whole can be sent on the I2C bus.

A unique transfer can be done via one of several methods. Each master could attempt to access a unique address. This address could either be a reserved address or simply an unused one. Because masters that address lower valued addresses, those which are numerically closer to 0x'00', win the arbitration this fact can be used to assign a type of priority scheme to the various competing masters. Note that in this method the IIC controller ends its

transfer in an error indication. If the IIC controller won the bus then the address non acknowledged bit in the extended status register is set. If the IIC controller lost the bus then the lost arbitration bit in the extended status register is set.

If it is not desirable to use an invalid or undefined address then the competing masters could each send a unique byte of data to the shared address(es). This technique works exactly as the unique address technique with one small exception. In this method because the arbitration takes place on the second byte (or third byte in 10-bit addressing mode) instead of the first byte, the time needed to assign a winning master is slightly longer, and as a result the bus throughput is slightly lower. Note that in this method the IIC controller ends its transfer in an error indication if it lost the bus, and the lost arbitration bit in the extended status register is set. If the core won the bus then the transfer ends with a normal ending status. Depending upon the needs of the application, the IIC controller might need to overwrite, or resend, the previously written unique data.

Another concern is when to know that enough time has elapsed after a reset to see activity on the I2C bus. One way to solve this problem is via the use of a super master, which can supervise the global operation of the I2C bus and ensure that it is operational. It can do this through the periodic issuance of a heartbeat transfer. This transfer is a harmless read or write operation that occurs periodically to ensure that the bus is not hung. If the periodicity of the heartbeat transfer is known then a master knows exactly the maximum time it must wait for activity after a reset happens.

33.4 Error Recovery

Recovering from an error can be a complex process. In many systems it is the most complex portion of the operating system, as well as the last portion to be implemented. This section provides information on how to analyze and respond to the various errors that can occur on the I2C bus. Most of these errors involve using the master's portion of the interface.

33.4.1 Time-Outs in Single Systems

Note: The following discussions assume that the software is in a time-out routine because the I2C bus and or devices are believed to have failed and are not simply busy and holding an I2C bus line low.

In a single master system the sole cause for a time-out is the hanging of a data transfer. Typically this involves another I2C bus device holding either the SCL or the SDA signal low. This state can be detected by reading the extended status register and seeing that the IIC controller is still in the master transfer state along with reading the direct control register and seeing that one or both of the I2C bus signals is low.

If both signals are being held low then the only thing that can be done is to wait for one of the lines to be released to the high state.

If the SDA line is being held low then the IIC controller should be held in reset by turning on the soft reset bit and leaving it on while recovery is attempted. The software should, via the direct control register, try setting the SCL signal as a low then waiting 1.3 µs (4.7 µs if the I2C bus's standard mode is being used). Then the software should set this line high and wait 0.6 µs (4.0 µs if the I2C bus's standard mode is used). The software should then check if the SDA line was released to high. If not then the software should repeat this at least nine times in an attempt to go through a one byte plus one acknowledge-bit cycle. Eventually this should free up the I2C bus unless the device holding the SDA line low has totally failed. Once the SDA line is returned to high, the software should remove the reset from the IIC controller by setting the soft reset bit to a '0'.

If the SCL line being is held low then the IIC controller should be held in reset by turning on the soft reset bit and leaving it on while recovery is attempted. The software should, via the direct control register, try setting the SDA signal as a low then waiting 1.3 µs (4.7 µs if the I2C bus's standard mode is being used). Then the software should set this line high and wait 0.6 µs (4.0 µs if the I2C bus's standard mode is being used). The software should then

check to see if the SCL line was released to high. If not, then the software should repeat this a number of times. This may free up the I2C bus unless the device holding the SCL line low has totally failed. This method is less effective since by definition the SDA line is allowed to change while the SCL line is low. Once the SCL line is returned to high the software should remove the reset from the IIC controller by setting the soft reset bit to a '0'.

If these techniques do not work then manual operator intervention is required.

33.4.2 Time-Outs in Multimaster Systems

In a multimaster system, in addition to failures caused by data transfer hangs, as noted previously, there are other failures which cause the bus to hang with no device being the active or competing master. In this case no master's time-out function would be hit and the bus would be hung forever. To handle this, one or more masters must have "super master" responsibility. It is the responsibility of the super master to determine the power-on or post-reset state of the I2C bus and to monitor global hangs on the I2C bus. Failures in these situations cause a time-out to occur. The former occurs after the IIC controller is reset or powered on; the latter occurs after a heartbeat check of the bus. A heartbeat check is one in which the super master attempts to read or write a harmless address on the I2C bus to ensure that the interface is still operational. If the bus is hung, then the heartbeat check fails in the same way as a hang in a normal data transfer.

In a post-reset time-out, the IIC controller has not been able to automatically determine the state of the I2C bus. To recover, the software must read the state of the SCL and SDA signals via the direct control register. In doing so the software can try to determine if the bus is in the inactive (or bus free), in use, or hung state. The hung state is the easiest one to identify. In this state, either the SCL, or the SDA or both the SCL and SDA signals are being held low. Recovery is identical to that for a hang in a data transfer. If both the SCL and the SDA lines are high then the I2C bus could be in either the inactive or hung state[1]. It might be possible to know the difference based upon the system architecture[2], but in general the super master cannot tell the difference between these states. To recover, the software should program the IIC controller to advance from the unknown I2C bus state, then perform a heartbeat transfer. If the bus was inactive, this transfer should be done without error. If the bus was hung then this transfer could either itself get hung, could end in an error, or it could cause the other I2C bus devices to see an illegal start/stop signal[3]. If the heartbeat transfer hangs, then the software should try to recover from a hung data transfer. If these techniques do not work then manual operator intervention is required.

Note 1: The bus could also be in use. But, by assumption, once a time-out is hit an in use bus is transformed into a hung bus. We define a bus that is not free and is not hung as in use.

Note 2: For example, once any I2C device is reset in the system, they are all reset due to the fact that they are all connected to the same master system reset signal. In this type of system, the super master would know that the bus is in the inactive state.

Note 3: The only way that the super master can sense an illegal Start/Stop in this case is if its heartbeat transfer causes a hung master to continue its transfer such that an illegal signal is created.

33.4.3 Illegal Start/Stop

The occurrence of an illegal signal during either a master or a slave transfer indicates a potential for corrupt information and should cause the device driver software to discard the results of the current operation. Preparation should be made to retry some number of operations to get back to whatever was the state at the time of the failure. For a master this means redoing some number of transfers. For a slave this means that some number of bytes may be rewritten or reread.

If the illegal signal happened during the master's transfer then the transfer was aborted and the master has moved into the busy I2C bus state. In the status register, the pending transfer bit is cleared to '0', the error bit is set to '1', and the halted or stopped is 0. The illegal start or stop in a master transfer bit in the extended status register is set. Any residual data in the master data buffer has been flushed. Prior to redoing the failed operation, the software is

need to reset and reinitialize the IIC controller. Consideration also needs to be given to the case where this core's slave might be addressed by another master. Care must be taken to handle a slave transfer just after the core's being reset and just prior to the resend of the aborted transfer.

If the illegal signal happened during a slave transfer then the transfer was aborted and the slave has moved into the busy I2C bus state. The software needs to clear the slave of status and to possibly discard any previous information associated with the failed transfer. Any residual data in the slave data buffer has been flushed. The software may also need to prepare for the retransfer of the failed operation. When this type of failure occurs, it is the responsibility of the master or super master to recover and resynchronize operations on the I2C bus. The software should leave the IIC controller in the busy I2C bus state as this aids the master in its failure recovery.

If the illegal signal happens when no transfer was being done, for example a large noise pulse happens, then the IIC controller's logic can most likely get hung in the slave transfer or busy I2C bus state. As no interrupt or status is generated this failure is seen as a time-out which has no active master. See "Time-Outs in Multimaster Systems" on page 1204. for how to recover from this type of error.

Note that this type of error indicates a systemic problem. Either an I2C bus master is malfunctioning, a master's software is mishandling post-reset operations, or the I2C bus signals are experiencing too much noise. Whatever the case, the problem needs to be identified and fixed.

Any status or interrupts that were set need to be cleared prior to the start of another transfer.

33.4.4 Address Non-Acknowledged

Once the device driver software has been debugged, purging it of nonexistent addresses, the remaining cause of this error is a device being too busy to respond to the requested transfer. A simple retry of the transfer is all that is needed to recover from this error. Note that excessive occurrences of this error could indicate that the master is operating at a rate which is too fast for the other devices in the system.

Any status or interrupts that were set need to be cleared prior to the start of another transfer.

33.4.5 Incomplete Transfer

Recovery from this error is the same as that for recovering from the address non-acknowledge error. The master's device driver should redo the transfer to either send or receive the remainder of the required data. It is possible that the device ended the transfer early due to either an overflow or underflow condition. It is further possible that as a result the I2C bus device is now in a busy state. In such a case, a fast retry of the incomplete transfer could result in a subsequent address non-acknowledge error. Prudent usage of either a query command to the busy slave (if one exists) or a wait period is advised in this case.

Any status or interrupts that were set need to be cleared prior to the start of another transfer.

33.4.6 Lost Arbitration

As has been stated previously, recovery from this error could be quite cumbersome. In general the IIC controller's master logic has to be programmed to redo the failed transfer. Included in the set of transfers that need to be redone are those transfers that were done just prior to the failed transfer and were used to put the slave into the state it was in when arbitration was lost.

Any status or interrupts that were set need to be cleared prior to the start of another transfer.

33.5 Slave Operation

Slave operations have a basic and inherent difference from master operations. In a master operation the master decides what to do and when to do it. A slave operation can occur at any time and is asynchronous to the actions in the rest of the slave. In general the slave has no idea what type of action is going to be requested by the master.

This unknown nature of a slave interface is not aided by the I2C bus architecture. The protocol provides for a single slave address[1] and only a read of data or a write of data indication. In such a system, provision must be made to establish what data is to be read or how to handle the data which is written. The design of the I2C bus requires that this provision be made at the system level. There are basically two ways in which to do this. One is a fixed typed of method and the other is more flexible or variable. In the fixed method, a certain amount of data is always written in a fixed order, and by definition the slave's device driver software knows what to do with it. Likewise in a fixed read, a certain amount of data is always read by the master and the order of the information is fixed and has a predefined meaning. Such a design is simple to implement but does provide slower system throughput. This is caused by the need to always transfer a fixed amount of information even when the master is only interested in a single byte. In a flexible, or variable method, a set of commands or sub addresses would be defined which then describe the next action to be done. Implementing such a variable method requires assigning meaning to the data which is written on the I2C bus immediately following the slave's address. Thus this data then becomes either a command (I wish to read data from location xyz) or is a sub address or pointer to another facility (I am accessing location xyz). Obviously a variable method of slave transfer is very powerful but is more complex. Not to be forgotten is the need to keep the master and slave in sync with what each other wants and is expecting. If secure or ensured transmissions are needed then some type of status exchange after the operation would have to be done. Remember that the I2C bus provides no protocol to ensure this. If this is needed in the system then the system software must implement it.

Note 1: A slave could occupy more than a single address on the I2C bus but this is not a recommended design. The bus only has seven or ten bits of address. Mapping a slave to more than one address in such a limited address space is not a wise choice. The IIC controller's slave only occupies one address on the bus.

When a Start signal is generated on the I2C bus, the IIC controller's slave logic, assuming it is in the free I2C bus state, advances to the slave transfer state. It is in this state that the IIC controller receives and decodes the incoming address. If the address does not match the value in the low slave address register (and the high slave address register if this is a 10-bit address operation) then the core does not acknowledge the address and advances to the busy I2C bus state. The core sits in this state until either a Stop or another Start is detected. A Stop advances the core to the free I2C bus state while a Start causes the controller to return to the slave transfer state. If the slave address matches the value in the core's slave address register(s) then a decision must be made according to the requested operation versus the state of the slave data buffer versus the settings in the mode control register. *Table 33-3* describes how the IIC controller's slave responds for the various possible states and program settings. If the slave enters the busy I2C bus state, as per this table, then the slave remains in this state until it detects a stop signal or a start signal.

Table 33-3. IIC Controller Operation (IICx_CNTL) When Addressed as an I2C Bus Slave

Enable Slave Mode	Enable Hold SCL (BCL)	I2C Bus Operation	Slave Data Buffer State	Resulting Action by the IIC Controller's Slave
0	X	X	X	Non-Ack the address, advance to busy I2C bus state
1	0	Write	Not Full	Ack the address advance to slave-selected state
1	0	Write	Full	Non-Ack the address, set status, advance to busy I2C bus state, set IRQ
1	0	Read	Not Empty	Ack the address, advance to slave-selected state.

Table 33-3. IIC Controller Operation (IICx_CNTL) When Addressed as an I2C Bus Slave

Enable Slave Mode	Enable Hold SCL (BCL)	I2C Bus Operation	Slave Data Buffer State	Resulting Action by the IIC Controller's Slave
1	0	Read	Empty	Non-Ack the address, set status, advance to busy I2C bus state, set IRQ
1	1	Write	Not Full	Ack the address, advance to slave-selected state.
1	1	Write	Full	Ack the address, set status, hold SCL low, advance to slave-selected state, set IRQ.
1	1	Read	Not Empty	Ack the address, advance to slave-selected state.
1	1	Read	Empty	Ack the address, set status, hold SCL low, advance to slave-selected state, set IRQ.

An 'X' is a 'don't care' value.
When state is set, an interupt, if enabled, is also set.
An 'Ack' refers to the acknowledge sent by the slave after the address.
'IRQ' means an interrupt request, or interrupt, is activated if enabled.

While in the slave-selected state, the slave performs the requested data transfer. Once all of the data is transferred the master issues either a stop or another start signal. Either of these signals ends the transfer and causes the slave to set status indicating that a slave transfer is complete. If the operation was a write then the slave write complete status is set and the slave data buffer contains some number of bytes. If the operation was a read then the slave read complete status is set and the slave data buffer is either empty or contains residual data that was not sent to the master. An interrupt is generated if so enabled in the interrupt enable register. If the transfer was ended by a stop signal then the IIC controller enters the free I2C bus state. If the transfer was ended by a start signal then the IIC controller remains in the slave-selected state until the address is decoded and then reacts according to *Table 33-3*.

During the transfer, that is, prior to the master's indicating the end of the transfer, it is possible for the slave to hit either an overflow or an underflow condition concerning the slave data buffer.

In an overflow condition, the master has written a byte of data and the slave data buffer is full. In this case one of two things can happen. If the enable hold SCL bit is set to a '0' then the I2C slave non-acknowledges this byte of data. This causes the master to end the transfer and thus generate I2C slave status and possibly an interrupt. If the enable hold SCL bit is set to a '1' then the I2C slave holds the SCL line low, pausing the I2C bus and thus preventing a byte from being received and overflowing the slave data buffer. The slave write needs service status is set along with a possible interrupt. In this last case the software needs to remove some or all of the bytes from the buffer and then clear the slave write needs service bit. At this point the slave releases the SCL line to high and permits the write operation to continue.

In an underflow condition, the master has read a byte of data and the slave data buffer is empty. In this case the slave holds the SCL signal low regardless of the setting in the enable hold SCL bit, because at this point in the transfer (other than sending junk data) the slave has no choice[1]. This causes the pausing of the I2C bus, thus

preventing an underflow condition in the slave data buffer. Slave read needs service status is set along with a possible interrupt. In this case the software needs to provide some data to the buffer and then clear the slave read needs service bit. At this point the slave releases the SCL line to high and permits the read operation to continue.

When slave status has been set to either a transfer complete or a needs service status indication, an interlocking condition is created inside the IIC controller's slave logic. This interlocking condition can, depending on how the core is programmed, either freeze the I2C bus or make the IIC controller busy for all transfers that address this core's slave. The interlocking condition is removed when the processor clears out the slave status.

When a transfer is completed either the slave write or read complete status is set in the extended control and slave status register. If this status is left set when another transfer is started (assuming that the IIC controller's slave is the one being addressed by this transfer) then the IIC controller can respond in one of two ways: If the enable hold SCL bit in the mode control register is a '1' then the IIC controller freezes the bus into a busy condition by holding the SCL signal low (see *Table 33-4* for more information). The I2C bus is held in this paused or busy condition until the slave complete status is cleared. If the enable hold SCL bit is a '0' then the IIC controller issues a non-acknowledge for this transfer (see *Table 33-4* for more information). The IIC controller continues to issue non-acknowledges until the slave complete status is cleared.[2]

Table 33-4. Needs Service Bit Status

Slave Data Buffer Status	Transfer Type	Result
Full	Write	Slave write needs service bit is set, along with a possible interrupt.
Empty	Read	Slave read needs service bit is set, along with a possible interrupt.

Note that a slave status interlocking condition can also occur following the initial write portion that precedes a 10-bit address read. Thus, for the 10-bit address read to occur, the processor has to clear a status that indicates zero bytes were written to the IIC controller's slave.

Note 1: The slave cannot use acknowledge to end a transfer because the master controls the acknowledge bit in a read operation. Thus once the slave acknowledges the address it is committed to send as many bytes as are requested by the master.

Note 2: This interlocked operation prevents a master from simply writing data to the IIC controller's slave and then trying to read it back without intervention from the processor inside the slave's system.

33.6 Interrupts

The IIC controller has two ways to handle the need for intervention. One way is for the processor to poll on the pending transfer or slave status set bits in the status register. The second method is to use the IIC controller's interrupt capability. Interrupts are globally and specifically controlled via the enable interrupt bit in the mode control register and the enable bits in the interrupt enable register. In addition, the shape of the interrupt signal can be selected to be a pulse or a constant level by setting the enable pulsed IRQ bit in the extended control and slave status register. Logic inside the macro ensures that the interrupt is activated after the state of the logic has been set. This ensures that an interrupt service routine can read stable values from the IIC controller. If the enable pulsed IRQ bit in the extended control and slave status register is set to a logic '1' then logic inside the IIC controller ensures that the interrupt signal is equal to a logic '1' for one and only one clock period. If the enable pulsed IRQ bit in the extended control and slave status register is set to a logic '0' then logic inside the IIC controller ensures that the interrupt signal stays equal to a logic '1' until the IRQ active bit in the status register is cleared to a logic '0'.

Since a master operation can have one interrupt and a slave operation can have two interrupts, the IIC controller must handle the case in which up to three interrupts have been queued up. The current interrupt is referred to as the active interrupt; the first interrupt in the queue is referred to as the on-deck interrupt, while the second queued interrupt is called the pending interrupt. Multiple interrupts are held in the queue until the active interrupt is cleared by writing a logic '1' to the IRQ active bit in the status register. Once the active interrupt is cleared, the on-deck interrupt becomes the active interrupt, and the pending interrupt becomes the on deck interrupt. Any queued interrupts can be seen by reading the Interrupt register.

Any status associated with any of the multiple interrupts is immediately set into its corresponding register. Thus, it is possible for an interrupt service routine to see the status for the current and the queued interrupts. This can occur if, for example, the interrupt routine is itself interrupted prior to reading any status in the IIC controller. If a second or third interrupt has occurred by the time the routine regains control, the status seen at this time includes status associated with all of the interrupting conditions. In this case, the routine cannot determine which was the first interrupting condition. They appear to have been merged together. Note that if the interrupt service routine is intelligent enough to handle and clear all conditions that are currently set in the status, as opposed to picking only one and servicing it, then the routine could see zero status for the subsequent interrupts.

A more typical situation is the case in which the interrupt routine has read the I2C status for the active interrupt and a second on-deck interrupt occurs. In this case, the status has changed after it was first read by the interrupt routine. Since status bits are cleared by writing a logic '1', no ill effects, such as lost status information, result from this case. For example, consider that the routine might have read 0x'10' in the extended control and slave status register when the first interrupt occurred. Then, sometime later, the routine wishes to clear this status by writing 0x'10' back to the register. If the slave write operation completes in between the time the register is read and written, the register contains a 0x'30' when the 0x'10' is written. Because writing a bit to logic '0' has no effect once it is logic '1', the slave write complete status is not lost. Note that if bits were cleared by writing a logic '0', then status would have been lost.

The IIC controller merges multiple interrupts into one interrupt under certain conditions, as shown in *Table 33-5*. This merge function is automatically done in the IIC controller's logic and is not under the program's control.

Table 33-5. Interrupt Merge Conditions

Active Or Queued Interrupt	Interrupt	Result
Slave Read Needs Service	Slave Read Complete	The two interrupts are merged into a single interrupt if the extended control and slave status register has not been read yet.
Slave Write Needs Service	Slave Write Complete	

33.7 Addressing

The IIC interfaces support 7-bit and 10-bit addressing for master and slave transfers. Addressing is described in detail in *IICx Low Master Address Register (IICx_LMADR)* on page 1214, *IICx High Master Address Register (IICx_HMADR)* on page 1215, *IICx Low Slave Address Register (IICx_LSADR)* on page 1222, and *IICx High Slave Address Register (IICx_HSADR)* on page 1223.

Descriptions of addressing modes and address formats follow.

33.7.1 Addressing Modes

For master transfers, the address mode (AMD) field of the IIC Control register (IICx_CNTL) controls whether 7-bit or 10-bit addresses are used. If IICx_CNTL[AMD] = 0, addresses contain 7 bits; if IICx_CNTL[AMD] = 1, addresses contain 10 bits.

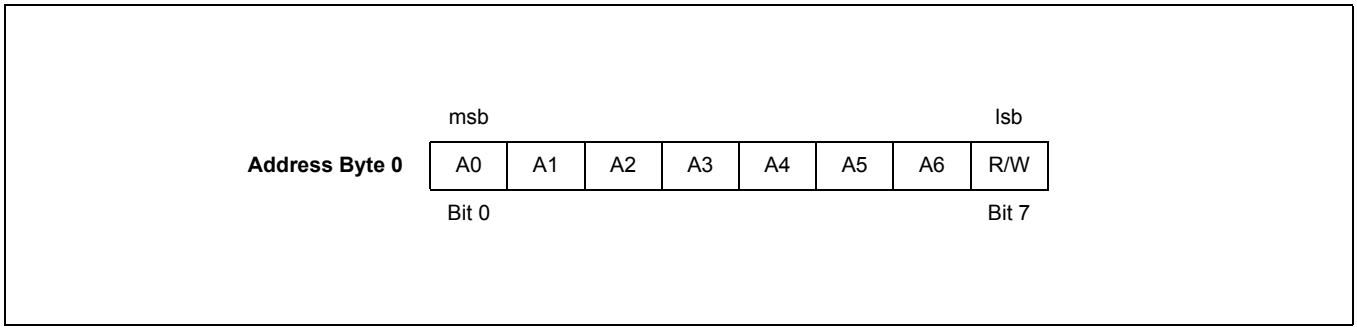
For slave transfers, the contents of the IICx High Slave Address register (IICx_HSADR) determines whether 7-bit or 10-bit addressing is used. If IICx_HSADR = 0b00000000, 7-bit addressing is used. If 10-bit addressing is to be used for slave transfers, IICx_HSADR = 0b11110*yy*x, where *yy* contains the high-order bits of the 10-bit address, and x is a "don't care."

Programming Note: For slave transfers, IICx_CNTL[AMD] does not control addressing mode.

33.7.2 Seven-Bit Addresses

Figure 33-3 illustrates a 7-bit address. For master transfers, the address bits 0 through 6 (A0:A6) are read from IICx_LMADR. For slave transfers, A0:A6 are read from IICx_LSADR. Bit 7 of address byte 0 contains a transfer type bit provided by the IIC interface.

Figure 33-3. 7-Bit Addressing



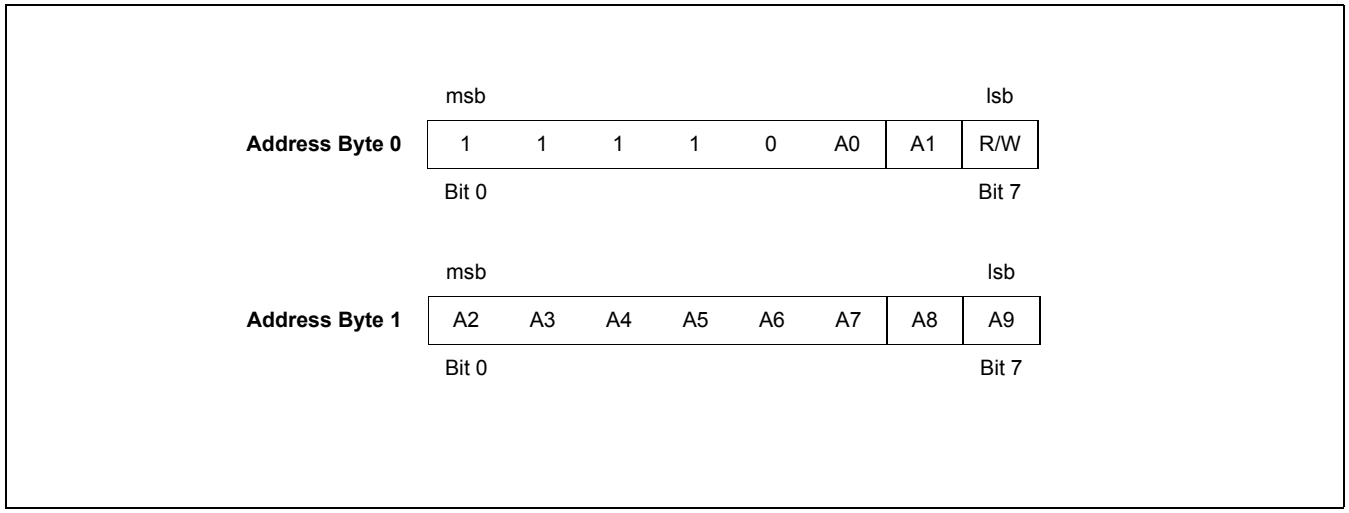
33.7.3 Ten-Bit Addresses

Figure 33-4 illustrates a 10-bit address. A0:A1 of address byte 0 are read from IICx_HMADR[A6:A7] (for master transfers) or IICx_HSADR[A6:A7] (for slave transfers). These are the two highest-order address bits transmitted on the IIC bus. Bit 7 of address byte 0 contains a transfer type bit provided by the IIC interface.

For 10-bit addressing for master or slave transfers, respectively, IICx_HMADR[A0:A4] and IICx_HSADR [A0:A4] must contain 0b11110.

The low-order byte of the 10-bit address, contained in A0:A7 of address byte 1, are read from IICx_LMADR or IICx_LSADR for master or slave transfers, respectively.

Figure 33-4. 10-Bit Addressing



33.8 IIC Registers

In the PPC460EX/EXr/GT there are two IICs, designated 0 and 1. In the following sections, the registers are specified with a generic name where x represents 0 or 1.

IIC registers are accessed at 0x4 EF60_0nYY where n = 7 for IIC0 and n = 8 for IIC1.

Table 33-6 lists the IICx registers. Descriptions of the registers, in the listed order, follow in *IIC Register Descriptions* on page 1213.

Table 33-6. IIC Registers

Mnemonic	Register	Address	Access	Effect of Reset	Bits	Page
IICx_MDBUF	Master Data Buffer	0x4 EF60 0n00	R/W	Cleared	8,16	1213
IICx_SDBUF	Slave Data Buffer	0x4 EF60 0n02	R/W	Cleared	8,16	1214
IICx_LMADR	Low Master Address	0x4 EF60 0n04	R/W	No	8	1214
IICx_HMADR	High Master Address	0x4 EF60 0n05	R/W	No	8	1215
IICx_CNTL	Control	0x4 EF60 0n06	R/W	Cleared	8	1216
IICx_MDCNTL	Mode Control	0x4 EF60 0n07	R/W	Cleared	8	1218
IICx_STS	Status	0x4 EF60 0n08	R/W	Cleared	8	1219
IICx_EXTSTS	Extended Status	0x4 EF60 0n09	R/W	Cleared	8	1220
IICx_LSADR	Low Slave Address	0x4 EF60 0n0A	R/W	No	8	1222
IICx_HSADR	High Slave Address	0x4 EF60 0n0B	R/W	No	8	1223
IICx_CLKDIV	Clock Divide	0x4 EF60 0n0C	R/W	Cleared	8	1223
IICx_INTRMSK	Interrupt Mask	0x4 EF60 0n0D	R/W	Cleared	8	1225
IICx_XFRCNT	Transfer Count	0x4 EF60 0n0E	R/W	Cleared	8	1226
IICx_XTCNTLSS	Extended Control and Slave Status	0x4 EF60 0n0F	R/W	Cleared	8	1227
IICx_DIRECTCNTL	Direct Control	0x4 EF60 0n10	R/W	0x0F	4	1228
IICx_INTR	Interrupt	0x4 EF60 0n11	R	Cleared	2	1229

33.9 IIC Register Descriptions

The following sections contains the bit definitions for the various registers in the IIC interface.

33.9.1 IICx Master Data Buffer (IICx_MDBUF)

The IICx Master Data Buffer (IICx_MDBUF) is a 1-B × 4-B first-in/first-out (FIFO) buffer. Data placed in IICx_MDBUF is written onto the IIC bus when performing a master write operation. Data received from the IIC bus is read from IICx_MDBUF when performing a master read operation.

Figure 33-5. IICx Master Data Buffer (IICx_MDBUF)

0		Data bit	
1		Data bit	
2		Data bit	
3		Data bit	
4		Data bit	
5		Data bit	
6		Data bit	
7		Data bit	

Half word reads from and writes to the IIC bus assume the MSB comes first. For half word writes, the first byte written to the IICx_MDBUF is the MSB, byte 0. The followed byte written is the LSB, byte 1. For half word reads, the first byte received is MSB, byte 0, and the following byte is LSB, byte 1.

If a byte is read from the IICx_MDBUF while the FIFO is empty, obsolete data is read. Check the master buffer status, IICx_STS[MDBS], before reading IICx_MDBUF.

If a byte is written to IICx_MDBUF while the FIFO is full, the byte is discarded and not placed into the FIFO. Check the master buffer full status, IICx_STS[MDBF], before writing IICx_MDBUF.

IICx_MDBUF is cleared (flushed and set to empty) whenever the IIC interface is reset, or IICx_MDCNTL[FMDB] = 1.

33.9.2 IICx Slave Data Buffer (IICx_SDBUF)

The IICx Slave Data Buffer (IICx_SDBUF) is a 1-byte 4-byte first-in/first-out (FIFO) buffer. The data contained in IICx_SDBUF is either received from the IIC bus when the IIC interface is addressed as a slave during a write, or is written on the IIC bus when the IIC interface is addressed as a slave during a read operation.

Figure 33-6. IICx Slave Data Buffer (IICx_SDBUF)

0		Data bit	
1		Data bit	
2		Data bit	
3		Data bit	
4		Data bit	
5		Data bit	
6		Data bit	
7		Data bit	

Half word reads from and writes to the IIC bus assume the MSB comes first. For half word writes, the first byte written to the IICx_SDBUF is the MSB, byte 0. The followed byte written is the LSB, byte 1. For half word reads, the first byte received is MSB, byte 0, and the following byte is LSB, byte 1.

If a byte is read from the IICx_SDBUF while the FIFO is empty, obsolete data is read. Check the slave buffer status, IICx_XTCNTLSS[SDBD], before reading IICx_SDBUF.

If a byte is written to IICx_SDBUF while the FIFO is full, the byte is discarded and not placed into the FIFO. Check the slave buffer full status, IICx_XTCNTLSS[SDBF], before writing IICx_SDBUF.

IICx_SDBUF is cleared (flushed and set to empty) whenever the IIC interface is reset, or IICx_MDCNTL[FSDB] = 1.

33.9.3 IICx Low Master Address Register (IICx_LMADR)

The IICx Low Master Address (IICx_LMADR) and IICx High Master Address Register (IICx_HMADR) form addresses that the IIC interface transmits on the IIC bus.

When in 7-bit address mode, IICx_CNTL[AMD] = 0, IICx_LMADR[A0:A6] is the address transmitted on the IIC bus; IICx_LMADR[A7] is a “don't care.” When in 10-bit address mode, IICx_CNTL[AMD] = 1, IICx_LMADR[A0:A7] is the second byte of the address transmitted on the IIC bus.

Figure 33-7. IICx Low Master Address Register (IICx_LMADR)

0	A0	Address bit 0	
1	A1	Address bit 1	
2	A2	Address bit 2	
3	A3	Address bit 3	
4	A4	Address bit 4	
5	A5	Address bit 5	
6	A6	Address bit 6	lsb for 7-bit addresses
7	A7	Address bit 7	lsb for 10-bit addresses; "don’t care" for 7-bit addresses

33.9.4 IICx High Master Address Register (IICx_HMADR)

IICx High Master Address Register (IICx_HMADR) provides the upper address bits in 10-bit addressing mode.

When in 10-bit address mode, IICx_CNTL[AMD] = 1, IICx_HMADR must be programmed to 0b1111 0yyz, where yy are the high-order bits of a 10-bit address and z is a don't care.

IICx_HMADR[A5:A6] are the two most significant bits of the 10-bit address. IICx_HMADR[A7] is a "don't care".

Figure 33-8. IICx High Master Address Register (IICx_HMADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	msb for 10-bit addresses
6	A6	Address bit 6	Next to msb for 10-bit addresses
7	A7	Address bit 7	Don't care for 10-bit addresses

33.9.5 IICx Control Register (IICx_CNTL)

The IICx Control Register (IICx_CNTL) starts and stops IIC interface master transfers on the IIC bus. When a transfer begins, the IIC interface uses the values in IICx_CNTL to determine the type and size of the transfer.

Programming Note: IICx_CNTL *must* be the last register programmed. Whenever IICx_CNTL[PT] = 1, the IIC interface attempts to perform the requested transfer using values set in other registers.

During and after transfers, the IICx_STS and IICx_EXTSTS registers can be read to determine the state of the IIC interface and the IIC bus.

Only IICx_CNTL[PT] is cleared when a requested master transfer is complete; the remaining bits are not affected.

Figure 33-9. IICx Control Register (IICx_CNTL)

0	HMT	Halt Master Transfer 0 Normal transfer operation. 1 Issue Stop signal on the IIC bus as soon as possible to halt master transfer.	If no transfer is in progress, no action is taken. IICx_CNTL[PT] needs not be set. If IICx_MDCNTL[EINT] = 1, an interrupt is generated. When set to a logic '1', the halt function is performed. If IIC controller was performing a requested master transfer, it issues the stop signal at the earliest possible point on the I2C bus. If the master is in either the wait or paused states then a stop signal is issued on the I2C bus. If no transfer was being done and if the master was not in either the wait or paused states, then no action is taken on the bus. In any case, an interrupt is activated if so enabled in the control registers. To halt a transfer, the pending transfer bit must be set to a logic '1'. **Note**: It is not recommended to halt a paused read operation as this could cause the master to get hung up trying to issue a Stop signal.
1	AMD	Addressing Mode 0 Use 7-bit addressing. 1 Use 10-bit addressing.	Does not affect slave transfers.
2:3	TCT	Transfer Count 00 Transfer one byte. 01 Transfer two bytes. 10 Transfer three bytes. 11 Transfer four bytes.	The count bits indicate how many bytes are in the requested master transfer. When all bytes have been successfully transferred, the requested transfer is considered complete. The precise action taken is dependent upon the programming in this register and in the mode control register.
4	BCL	Hold Bus Ownership 0 Hold Bus Ownership Disabled 1 Hold Bus Ownership Enabled	When set to a logic '1', the current requested master transfer ends with no Stop signal issued on the I2C bus. The IIC controller enters into a Wait state until the next master transfer is requested. Ownership of the I2C bus is held by this controller's master while it is in the wait state. When the next master transfer is requested, the controller exits the wait state and the transfer is started using a repeated start function on the I2C bus.
5	CHT	Chain Transfer 0 Transfer is only or last transfer. 1 Transfer is one of a sequence of transfers (but not last in sequence).	When set to a logic '1', the requested master transfer is one in a sequence of transfers. Completion of the requested transfer only indicates that this piece of the transfer is complete. The next action taken on the I2C bus is under the control of the program. When set to a logic '0', the requested master transfer is either the only transfer to be performed at this time or it is the last transfer in a sequence of transfers. In either case, completion of the requested transfer causes a stop condition to be sent on the I2C bus.

6	RW	Read/Write 0 Transfer is a write. 1 Transfer is a read.	
7	PT	Pending Transfer 0 Most recent requested transfer is complete. 1 Start transfer if bus is free.	When set to a logic '1', a master transfer is started, if the I2C bus is free. The type of requested transfer is specified by the other bits in this register. If the I2C bus is busy, then I2C waits for the bus to become free and then starts the requested transfer. This bit is cleared when the requested transfer is completed. When the transfer is complete, an interrupt, if enabled in the mode control register, is activated.

Table 33-7 summarizes IIC interface operation for settings of IICx_CNTL[HMT, BCL, CHT, PT]. x represents a don't care in the BCL, CHT and PT columns.

Table 33-7. IIC Response to IICx_CNTL Field Settings

IICx_CNTL Fields				Resulting Action on IIC Bus and Inside IIC Interface
HMT	BCL	CHT	PT	
0	x	x	0	No action taken
0	0	1	1	Start, Transfer, ACK on last byte, Pause
0	0	0	1	Start, Transfer, NACK on last byte, Stop
1	x	x	x	NACK on current byte, Stop
0	1	x	1	Start, Transfer, NACK on last byte, Wait

Settings of IICx_CNTL[HMT, BCL, CHT, PT] result in the following actions.

Start IIC Start condition generated, if the IIC interface was stopped or waiting.

Stop IIC Stop condition generated; IIC interface enters the Stop condition.

ACK IIC Acknowledge condition generated.

NACK IIC Not Acknowledge condition generated, if performing a read.

Transfer Requested bytes are transferred.

Pause IIC interface enters the Pause state.

Wait IIC interface enters the Wait state.

IICx_CNTL[HMT] overrides IICx_CNTL[BCL, CHT].

IICx_CNTL[BCL, PT] overrides IICx_CNTL[CHT].

33.9.6 IICx Mode Control Register (IICx_MDCNTL)

The IICx Mode Control Register (IICx_MDCNTL) sets the major modes of operation on the IIC bus. In addition, IICx_MDCNTL can force the data buffers into the empty state.

In typical applications, IICx_MDCNTL is configured once, during software initialization. Applications providing complex error handling may reconfigure this register more often.

> **Programming Note:** IICx_CLKDIV *must* be initialized before IICx_MDCNTL. IICx_LSADR and IICx_HSADR should also be configured before IICx_MDCNTL.

Note that the IIC hardware does not implement time-out functions on the IIC bus. Such functions must be implemented, in software, by setting IICx_CNTL[HMT] = 1, or setting IICx_XTCNTLSS[SRST] = 1.

Regarding IICx_MDCNTL[HSCL], a "slave not ready" condition occurs during a slave receive operation, if there is no space in the slave data buffer at the start of a write operation, or if the slave data buffer fills during the write. In a slave transmit operation, a slave not ready condition occurs if there is no data in the slave data buffer at the start of a read operation, or if the slave data buffer becomes empty during the read.

Using IICx_MDCNTL[HSCL] to handle slave not ready conditions can affect system performance. A slave holding the IICSCL signal low guarantees data delivery to or from the requesting master, but prevents other masters from performing transfers over the IIC bus. There is no general rule for handling slave not ready conditions; each system has its own requirements.

Figure 33-10. IICx Mode Control Register (IICx_MDCNTL)

0	FSDB	Flush Slave Data Buffer 0 Normal operation 1 Set slave data buffer to empty.	Cleared after buffer is emptied.
1	FMDB	Flush Master Data Buffer 0 Normal operation 1 Set master data buffer to empty.	Cleared after buffer is emptied.
2	EGC	Enable General Call 0 Ignore general call on IIC bus. 1 Respond to general call on IIC bus.	IICx_MDCNTL[ESM] overrides this field; if IICx_MDCNTL[ESM] = 1, a general call is ignored.
3	FSM	Fast/Standard Mode 0 IIC transfers run at 100 kHz (standard mode). 1 IIC transfers run at 400 kHz (fast mode).	
4	ESM	Enable Slave Mode 0 Slave transfers are ignored. 1 Slave transfers are enabled.	Program IICx_LSADR and IICx_HSADR before setting this field.
5	EINT	Enable Interrupt 0 Interrupts are disabled. 1 Enables interrupts for interrupts enabled in IICx_NTRMSK.	
6	EUBS	Exit Unknown IIC Bus State 0 Normal operation. 1 IIC bus control state machine exits unknown bus state, if in an unknown state.	If the IIC bus control state machine is in a known state, setting IICx_MDCNTL[EUBS] = 1 has no effect.
7	HSCL	Hold IIC Serial Clock Low 0 If slave is not ready, issue a NACK in response to slave transfer request. 1 If slave is not ready, hold the IICSCL signal low until slave is ready.	This field is used only when in slave mode.

33.9.7 IICx Status Register (IICx_STS)

The IICx Status register (IICx_STS) contains the state of the IIC interface and the status of any previously requested master transfers.

During and after transfers, software can read the IICx_STS and IICx_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

Programming Note: IICx_STS should be the first register read by an interrupt or error handler routine. IICx_STS can also be read in a polling loop if the IIC interrupts are not used.

All IICx_STS bit fields except for IICx_STS[SSS] must be cleared before requesting another master transfer. IICx_STS[SSS] is not affected by master transactions.

Figure 33-11. IICx Status Register (IICx_STS)

0	SSS	Slave Status Set 0 No slave operations are in progress. 1 Slave operation is in progress.	Read-only; this field is set when any of the following fields are set: IICx_XTCNTLSS[SRC, SRS, SWC, SWS].
1	SLPR	Sleep Request 0 Normal operation. 1 Sleep mode (SDR0_ER[IIC] = 1).	Read-only. The IIC interface is awakened when a start signal is detected on the IIC bus or when the SDR0_ER[IICx] is cleared.
2	MDBS	Master Data Buffer Status 0 Master data buffer is empty. 1 Master data buffer contains data.	Read-only.
3	MDBF	Master Data Buffer Full 0 Master data buffer is not full. 1 Master data buffer is full.	Read-only.
4	SCMP	Stop Complete 0 No request to halt transfer, or master data transfer, is complete. 1 Request to halt transfer, or master data transfer, is complete.	To clear IICx_STS[SCMP], set IICx_STS[SCMP] = 1.
5	ERR	Error 0 No error has occurred. 1 One of the following fields is set: IICx_EXTSTS[LA, ICT, XFRA] = 1.	Read-only.
6	IRQA	IRQ Active 0 No IIC interrupt has been sent to the universal interrupt controller (UIC). 1 An IIC interrupt has been sent to the UIC.	To clear IICx_STS[IRQA], set IICx_STS[IRQA] = 1. If IICx_MDCNTL[EINT] = 0, then IICx_STS[IRQA] is not set.
7	PT	Pending Transfer 0 No transfer is pending, or transfer is in progress. 1 Transfer is pending.	Read-only.

The Error and Pending Transfer, IICx_STS[ERR, PT], bit fields indicate the success or failure of the requested transfer. *Table 33-8* lists the transfer status for all combinations of the IICx_STS[ERR,PT] bit fields.

Table 33-8. IICx_STS[ERR, PT] Decoding

ERR	PT	Status
0	0	Requested transfer completed without errors
0	1	Requested transfer is in progress; no errors were detected
1	0	Requested transfer is complete, but not all data was transferred
1	1	Requested transfer is in progress; but an error was detected

Programming Note: No action regarding a master transfer should be taken unless all pending transfers have completed, IICx_STS[PT] = 0.

If an error requires the IIC interface to send a Stop, the Stop Complete bit field is set, IICx_STS[SCMP] = 1. Note that slave operations should be serviced regardless of the state of a requested master transfer.

IIC1_MDCNTL[EUBS] must be set after a reset before IIC interface can be placed in sleep mode. The IIC interface is placed in sleep mode by setting the CPM0_ER[IICx] via software. Awaking the IIC interface is possible directly through software by clearing the CPM0_ER[IICx] or indirectly by detecting a Start condition on the IIC bus. When a Start condition is detected, the IIC interface is awakened, clearing the CPM0_ER[IICx] and the IICx_STS[SLPR].

The IICx_STS[MDBS, MDBF] contain the current status of the Master Data Buffer, IICx_MDBUF. When the IICx_MDBUF contains data, IICx_STS[MDBS] is set. When the IICx_MDBUF is full, IICx_STS[MDBF] is set.

The state of the IICx_MDBUF is not instantly recorded by the IICx_STS[MDBS, MDBF]. The delay depends on the size of the buffer access. For half word accesses, these fields are valid on the third OPB clock following the transfer. For byte accesses, these fields are valid on the second OPB clock following the transfer.

33.9.8 IICx Extended Status Register (IICx_EXTSTS)

The IICx Extended Status register (IICx_EXTSTS) reports additional IIC status.

During and after transfers, software can read the IICx_STS and IICx_EXTSTS registers to determine the state of the IIC interface and the IIC bus.

Figure 33-12. IICx Extended Status Register (IICx_EXTSTS)

0	IRQP	IRQ Pending 0 No IRQ is pending. 1 An IRQ is active, another IRQ is on-deck, and another interrupt-generating condition has occurred.	• IICx_EXTSTS[IRQP] might be set momentarily while an IRQ moves from the Pending to the On-deck state. • An interrupt remains pending, IICx_EXTSTS[IRQP] = 1, until the current on-deck interrupt becomes active, IICx_STS[IRQD]=0 and IICx_STS[IRQA] = 1. • Writing 1 to IICx_EXTSTS[IRQP] clears the field. • When the IIC interrupt is disabled, IICx_MDCNTL[IRQP] = 0, IICx_EXTSTS[IRQP] should be ignored.

1:3	BCS	Bus Control State 000 Unused; if this value is read an error occurred. 001 Slave-selected state; the IIC interface has detected and decoded a slave transfer request on the IIC bus. 010 Slave Transfer state; the IIC interface has detected but has not decoded a slave transfer request on the IIC bus. 011 Master Transfer state; entered after a master transfer request has started on the IIC bus. 100 Free Bus state; the bus is free and no transfer request is pending. 101 Busy Bus state; the bus is busy. 110 Unknown state; value after IIC reset. 111 Unused; if this value is read an error occurred.	Read-only.
4	IRQD	IRQ On-Deck 0 No IRQ is on-deck. 1 An interrupt is active, and another interrupt-generating condition has occurred.	• IICx_EXTSTS[IRQD] might be set momentarily while an IRQ moves from the On-deck to the Active state. • An interrupt remains on-deck, IICx_EXTSTS[IRQD = 1, until the current active interrupt is no longer active, IICx_STS[IRQA] = 0. • If IICx_EXTSTS[IRQP] = 1, IICx_EXTSTS[IRQD] is set on the next OPB clock. • Writing 1 to IICx_EXTSTS[IRQD] clears the field. • When the IIC interrupt is disabled, IICx_MDCNTL[IRQP]=0, IICx_EXTSTS[IRQD] should be ignored.
5	LA	Lost Arbitration 0 Normal operation. 1 Loss of arbitration has ended the requested master transfer.	• If arbitration is lost, any requested master transaction may have terminated prematurely. Read data may be incomplete and not all write data may have been written. • If arbitration is lost during a repeat start, the master may not own the IIC bus.
6	ICT	Incomplete Transfer 0 Normal operation. 1 Some of the bytes of the requested master transfer were not transferred.	For an incomplete transfer, read the transfer count, IICx_XFRCNT, to determine how bytes were transferred.
7	XFRA	Transfer Aborted 0 No transfer is pending, or transfer is in progress. 1 A requested master transfer was aborted by a NACK during the transfer of the address byte, or was aborted because arbitration was lost. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.	Transfer aborted. When set to a 1, a requested master transfer was aborted by a NOT acknowledge during the transfer of the address byte. It is also set to a 1 when a requested master transfer loses data. Lost arbitration can be caused by the loss of data during the transfer of the second or subsequent data byte.

IICx_EXTSTS[IRQP, IRQD] and IICx_STS[IRQA] provide a FIFO for storing interrupts. A new interrupt is considered pending, and remains pending while an on-deck interrupt is present. Once the on-deck interrupt becomes active, the pending interrupt moves on-deck, and remains on-deck until there is no active interrupt. When the active interrupt is cleared, the on-deck (initially pending) interrupt becomes active.

Programming Note: An active interrupt remains active until software clears it.

IICx_EXTSTS[BCS] indicates the state of the IIC interface. The field is read-only.

IICx_EXTSTS[LA, ICT, XFRA] are cleared when IICx_EXTSTS[XFRA] = 1.

Writing 1 to IICx_EXTSTS[IRQP, IRQD, LA, ICT, XFRA] clears these fields.

33.9.9 IICx Low Slave Address Register (IICx_LSADR)

The IICx Low Slave Address Register (IICx_LSADR) and IICx_ High Slave Address Register (IICx_HSADR) program the slave address of the IIC interface. IICx_HSADR is used only for 10-bit addressing, and is not programmed in 7-bit addressing mode.

When 7-bit addressing is used, IICx_LSADR is written with the slave address; IICx_HSADR must be written with zeros. For 7-bit addressing, IICx_LSADR[A0:A6] contain the address transmitted on the IIC bus; IICx_LSADR[A7] is a "don't care.".

When 10-bit addressing is used, IICx_LSADR[A0:A7] contain the second address byte transmitted on the IIC bus.

Figure 33-13. IICx Low Slave Address Register (IICx_LSADR)

0	A0	Address bit 0	
1	A1	Address bit 1	
2	A2	Address bit 2	
3	A3	Address bit 3	
4	A4	Address bit 4	
5	A5	Address bit 5	
6	A6	Address bit 6	lsb for 7-bit addresses
7	A7	Address bit 7	lsb for 10-bit addresses; "don't care" for 7-bit addresses

33.9.10 IICx High Slave Address Register (IICx_HSADR)

For 7-bit addressing, set IICx High Slave Address Register (IICx_HSADR) to 0.

To enable 10-bit slave addressing, IICx_HSADR must be programmed to 0b1111 0*yyx*, where *yy* are the high-order bits of a 10-bit address and *x* is a "don't care.".

Programming Note: IICx_HSADR is used only for 10-bit addressing, and should be set to 0 for 7-bit addressing mode.

Thus, in 10-bit address mode, IICx_HSADR[A6:A7] contain the two highest -order bits of the 10-bit address; IICx_HSADR[A7] is a "don't care.". IICx_LSADR contains the low-order byte of the 10-bit address.

Figure 33-14. IICx High Slave Address Register (IICx_HSADR)

0	A0	Address bit 0	1 for 10-bit addresses
1	A1	Address bit 1	1 for 10-bit addresses
2	A2	Address bit 2	1 for 10-bit addresses
3	A3	Address bit 3	1 for 10-bit addresses
4	A4	Address bit 4	0 for 10-bit addresses
5	A5	Address bit 5	msb for 10-bit addresses
6	A6	Address bit 6	Next to msb for 10-bit addresses
7	A7	Address bit 7	Don't care for 10-bit addresses

Thus, in 10-bit address mode, bits 0:6 are used to decode the first address byte that was transmitted on the IIC bus, and bit 7 is in a "don't care.".

33.9.11 IICx Clock Divide Register (IICx_CLKDIV)

The IICx Clock Divide Register (IICx_CLKDIV) establishes a reference between the OPB clock and the IIC bus serial clock.

Programming Note: IICx_CLKDIV must be initialized before IICx_MDCTRL. Until IICx_CLKDIV is initialized, all IIC bus activity is ignored.

Figure 33-15. IICx Clock Divide Register (IICx_CLKDIV)

0	DIV0	Divisor bit 0	
1	DIV1	Divisor bit 1	
2	DIV2	Divisor bit 2	
3	DIV3	Divisor bit 3	
4	DIV4	Divisor bit 4	
5	DIV5	Divisor bit 5	
6	DIV6	Divisor bit 6	
7	DIV7	Divisor bit 7	

IICx_CLKDIV divides PPC460EX/EXr/GT's on-chip peripheral bus (OPB) clock to form the base clock for the IIC bus.

Table 33-9 lists the divisor values for several OPB frequency ranges. These divisor values apply for standard and fast mode. Select the divisor value by matching the OPB clock frequency to the corresponding frequency range in *Table 33-9*. For example, if the OPB clock frequency is 50MHz, select a divisor value of 0x4.

Table 33-9. IICx Clock Divide Programming

OPB Frequency Range (MHz)	Divisor Value
20	0x1
$20 < f \leq 30$	0x2
$30 < f \leq 40$	0x3
$40 < f \leq 50$	0x4
$50 < f \leq 60$	0x5
$60 < f \leq 70$	0x6
$70 < f \leq 80$	0x7
$80 < f \leq 90$	0x8
$90 < f \leq 100$	0x9
$100 < f \leq 110$	0xA
$110 < f \leq 120$	0xB
$120 < f \leq 130$	0xC
$130 < f \leq 140$	0xD
$140 < f \leq 150$	0xE

33.9.12 IICx Interrupt Mask Register (IICx_INTRMSK)

The IICx Interrupt Mask Register (IICx_INTRMSK) specifies which conditions can generate an IIC interrupt when the IIC interrupt is enabled, IICx_MDCNTL[EINT] = 1.

Figure 33-16. IICx Interrupt Mask Register (IICx_INTRMSK)

0	EIRC	Enable IRQ on Slave Read Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave read on the IIC bus. IICx_XTCNTLSS[SRC] = 1 indicates a Slave Read Complete.
1	EIRS	Enable IRQ on Slave Read Needs Service 0 Disable 1 Enable	The interrupt is activated upon receipt of a slave read on the IIC bus and the slave buffer was empty or went empty and more data was requested on the IIC bus. **Note:** IICx_XTCNTLSS[SRS] = 1 indicates a Slave Read Needs Service.
2	EIWC	Enable IRQ on Slave Write Complete 0 Disable 1 Enable	The interrupt is activated upon receipt of a Stop during a slave write on the IIC bus. **Note:** IICx_XTCNTLSS[SWC] = 1 indicates a Slave Write Compete.
3	EIWS	Enable IRQ on Slave Write Needs Service 0 Disable 1 Enable	The interrupt is activated when the slave buffer becomes full during a slave write on the IIC bus. **Note:** IICx_XTCNTLSS[SWS] = 1 indicates a Slave Write Needs Service.
4	EIHE	Enable IRQ on Halt Executed 0 Disable 1 Enable	
5	EIIC	Enable IRQ on Incomplete Transfer 0 Disable 1 Enable	
6	EITA	Enable IRQ on Transfer Aborted 0 Disable 1 Enable	
7	EIMTC	Enable IRQ on Requested Master Transfer Complete 0 Disable 1 Enable	

33.9.13 IICx Transfer Count Register (IICx_XFRCNT)

The IICx Transfer Count Register (IICx_XFRCNT) reports the number of bytes transferred on the IIC bus during a master or a slave operation.

Figure 33-17. IICx Transfer Count Register (IICx_XFRCNT)

0		Reserved	
1:3	STC	Slave Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved	
4		Reserved	
5:7	MTC	Master Transfer Count 000 0 bytes transferred 001 1 byte transferred 010 2 bytes transferred 011 3 bytes transferred 100 4 bytes transferred 101 Reserved 110 Reserved 111 Reserved	

IICx_XFRCNT[MTC] is cleared when there is a pending transfer, IICx_CNTL[PT] = 1.

IICx_XFRCNT[STC] is cleared when:

- A slave operation starts on the IIC bus
- Software indicates the slave does not need service by clearing IICx_XTCNTLSS[SRS] or IICx_XTCNTLSS[SWS].

33.9.14 IICx Extended Control and Slave Status Register (IICx_XTCNTLSS)

The IICx Extended Control and Slave Status Register (IICx_XTCNTLSS) provides additional control of IIC interface functions and reports the status of slave operations.

Figure 33-18. IICx Extended Control and Slave Status Register (IICx_XTCNTLSS)

0	SRC	Slave Read Complete 0 Normal operation, or IICx_MDCNTL[HSCL] = 0, IICx_SDBUF is empty, and a read operation is in progress. 1 A NACK or Stop condition was received over the IIC bus, or a repeated Start condition ended a read operation.	Check whether the read operation emptied IICx_SDBUF.
1	SRS	Slave Read Needs Service 0 Normal operation or slave read does not need service. 1 IICx_SDBUF is empty, and a read operation was requested on the IIC bus. The set condition may also indicate that IICx_SDBUF is empty due to a slave read and additional data is requested by the master.	1. If IICx_MDCNTL[HSCL]=0 and IICx_SDBUF contains no data, the slave will issue a NACK and IICx_XTCNTLSS[SRS] is set. 2. If ICx_MDCNTL[HSCL]=0, and IICx_SDBUF contains data, the slave will send the data. IICx_XTCNTLSS[SRS] is not set unless the master request additional data. 3. If IICx_MDCNTL[HSCL]=0, and IICx_SDBUF contains no data, the slave will hold IICSCL low to indicate the slave is busy. IICx_XTCNTLSS[SRS] is set until the IICx_SDBUF is filled. Once filled, IIC-SCL is released, IICx_XTCNTLSS[SRS] is cleared, and the slave sends the data. 4. If IICx_MDCNTL[HSCL]=1, and IICx_SDBUF contains data, the slave will send the data. IICx_XTCNTLSS[SRS] is not set unless the master requests additional data.
2	SWC	Slave Write Complete 0 Normal operation or slave write in progress. 1 A Stop signal was received during a write operation, or a repeated Start condition ended a write operation.	
3	SWS	Slave Write Needs Service 0 Normal operation or slave write does not need service. 1 IICx_SDBUF is full during a slave write.	1. If IICx_MDCNTL[HSCL] = 1 and IICx_SDBUF is full, the slave will hold IICSCL low to indicate the slave is busy. IICx_XTCNTLSS[SWS] is set until IICx_SDBUF is empty. Once empty, IICSCL is released, IICx_XTCNTLSS[SWS] is cleared, and the slave receives the data. 2. If IICx_MDCNTL[HSCL] = 0 and IICx_SDBUF is full, the slave will issue a NACK and IICx_XTCNTLSS[SWS] is set.
4	SDBD	Slave Data Buffer Has Data 0 IICx_SDBUF is empty 1 IICx_SDBUF contains data	Read-only
5	SDBF	Slave Data Buffer Full 0 IICx_SDBUF is not full 1 IICx_SDBUF is full	Read-only
6		Reserved	
7	SRST	Soft Reset 0 Normal operation 1 Soft reset	

Writing a 1 to IICx_XTCNTLSS[SRC, SRS, SWC, SWS] clears these fields.

The IICx_XTCNTLSS[SBSS, SDBF] contain the current status of the Slave Data Buffer, IICx_SDBUF. When the IICx_SDBUF contains data, IICx_XTCNTLSS[SDBD] is set. When the IICx_SDBF is full, IICx_XTCNTLSS[SDBF] is set.

The state of the IICx_SDBUF is not instantly recorded by the IICx_XTCNTL[SDBD, SDBF]. The delay depends on the size of the buffer access. For half-word accesses, these fields are valid on the third OPB clock following the transfer. For byte accesses, these fields are valid on the second OPB clock following the transfer.

If any of the following fields: IICx_XTCNTLSS[SRC, SRS, SWC, SWS] = 1 and IICx_MDCNTL[HSCL] = 0; no new slave operations will be accepted over the IIC bus. A NACK is issued until IICx_XTCNTLSS[SRS] or IICx_XTCNTLSS[SRS], and IICx_XTCNTLSS[SRS] or IICx_XTCNTLSS[SRS], are cleared.

Soft reset, IICx_XTCNTLSS[SRST], provides a last means of recovery from IIC interface or IIC bus failure. Once enabled, soft reset completely resets the IIC interface. All IIC registers are affected. All transmissions from the IIC interface are terminated. Enabling soft reset during an IIC transmission may improperly terminate the transmission and hang the IIC bus.

33.9.15 IICx Direct Control Register (IICx_DIRECTCNTL)

The IICx Direct Control Register (IICx_DIRECTCNTL), which controls and monitors the IIC serial clock (IICSCL) and serial data (IICSDA) signal, is used for error recovery when a malfunction is detected on the IIC interface.

Figure 33-19. IICx Direct Control Register (IICx_DIRECTCNTL)

0:3		Reserved	
4	SDAC	IICSDA Output Control Directly controls the IICSDA output. 0 IICSDA is a logic 0 1 IICSDA is a logic 1	
5	SCLC	IICSCL Output Control Directly controls the IICSCL output 0 IICSCL is a logic 0 1 IICSCL is a logic 1	
6	MSDA	Monitor IICSDA Used to monitor the IICSDA input 0 IICSDA is a logic 0 1 IICSDA is a logic 1	Read-only
7	MSCL	Monitor IICSCL. Used to monitor the IICSCL input. 0 IICSCL is a logic 0 1 IICSCL is a logic 1	Read-only

IICx_DIRECTCNTL[SDAC, SCLC] can be written to control the IICSDA and IICSCL signals. When controlling the IICSDA and IICSCL signals directly, the IIC controller must be placed in the reset state, IICx_XTCNTL[SRST] = 1.

IICx_DIRECTCNTL[MSDA, MSCL] are used to verify that IICx_DIRECTCNTL[SDAC, SCLC] were written successfully, and that the IICSCL signals can be controlled. If IICx_DIRECTCNTL[MSDA, MSCL] do not correspond to IICx_DIRECTCNTL[SDAC, SCLC], respectively, toggle IICSCL repeatedly to regain control.

IICx_DIRECTCNTL[SDAC, SCLC, MDSA, MSCL] = 1 after a chip or system reset. A Soft Reset, IICx_XTCNTLSS[SRST] = 1, does not affect the state of IICx_DIRECRCNTL.

33.9.16 IICx Interrupt Register (IICx_INTR)

This register is used for diagnostic purposes, providing information on queued or pending interrupts. This register is not visible on the OPB if Enable_32byte_window is false.

Bit 1 is cleared when the on-deck interrupt is cleared and the pending IRQ has become the on-deck IRQ. Bit 0 is cleared when the active interrupt is cleared and the on-deck IRQ has become the active IRQ. Both bits are cleared by a hard or a soft reset.

The pending and on-deck interrupts, along with the active interrupt in the status register, form a miniature FIFO for storing the interrupts. A new interrupt is first set into the pending state. It will stay pending as long as an on-deck interrupt is present. Once the on-deck interrupt is cleared, or if none was present at the time the new interrupt occurred, the pending interrupt is moved into the on-deck state. An on-deck interrupt remains in the on-deck state as long as an active interrupt is present. Once the active interrupt is cleared, or if none was present at the time the pending interrupt went to the on-deck state, the on-deck interrupt is moved into the active state. Note that an active interrupt remains in the active state until it is cleared by the program.

Figure 33-20. IICx Interrupt Register (IICx_INTR)

0	IRQD	IRQ On-Deck Set when an IRQ is still active and another interrupting condition was generated.	This bit might also be momentarily set while a new interrupting condition moves from the on-deck to the active state. When IRQ active is set to 0, an on-deck IRQ will cause IRQ active to be set 1, and IRQ on-deck will be cleared. If there is a pending IRQ, then IRQ on-deck will be set to 1 on the next system clock. The on-deck interrupt can be cleared by writing 1 to this bit. If interrupts are disabled, then bit 2 in the Mode Control Register is 0, and this bit will not be set.
1	IRQP	IRQ Pending Set when an IRQ is still active, an IRQ is on-deck, and another interrupting condition was generated.	This bit might also be momentarily set while a new interrupting condition moves from the pending to the on deck state. When IRQ on-deck is set to 0, a pending IRQ will cause IRQ on-deck to be set to 1, and IRQ pending will be cleared. The pending interrupt can be cleared by writing 1 to this bit. If interrupts are disabled, then bit 2 in the Mode Control Register is 0, and this bit will not be set.
2:7		Reserved	

33.10 Interrupt Handling

Service request on the IIC interface can be monitored by polling the IICx_STS[SSS, PT] or by using the IIC interrupt to the UIC. When the IICx_MDCNTL[EINT] is enabled, the IIC interface generates an interrupt to the UIC if an unmasked IIC interrupt condition occurs. The interrupt is recorded by the UIC if UIC0_ER[IICx] is enabled. The conditions that generate an IIC interrupt to the UIC are determined by the interrupt mask settings in IICx_INTMSK.

The IIC interface can queue up to three interrupts since there is one interrupt for master operations and two for slave operations. The current interrupt is referred to as the *active* interrupt. The first interrupt in the queue is the on-deck interrupt; the second queued interrupt is called the pending interrupt. The queue holds multiple interrupts until the active interrupt is cleared by writing a 1 to IICx_STS[IRQA]. When an active interrupt is cleared, the on-deck interrupt becomes the active interrupt and the pending interrupt becomes the on-deck interrupt.

When multiple interrupts occur, the status of the active interrupt and the queued interrupt merge making it impossible to determine which of the conditions originated the active interrupt. An interrupt handle should therefore save the contents of the status registers (IICx_STS and IICx_XTCTLSS) and handle all conditions set. Once handled, the status conditions can be cleared by overwriting the status registers with their saved content. If another IIC interrupt condition occurs before clearing the status register, the status bit of this condition is preserved since status bits are cleared when written with a 1.

When multiple interrupts occur, the status of the active interrupt and the queued interrupt merge making it impossible to determine which of the conditions originated the active interrupt. An interrupt handle should therefore save the contents of the status registers (IICx_STS and IICx_XTCTLSS) and handle all conditions set. Once handled, the status conditions can be cleared by overwriting the status registers with their saved content. If another IIC interrupt condition occurs before clearing the status register, the status bit of this condition is preserved since status bits are cleared when written with a 1.

Under certain conditions, the IIC interface merges slave read (write) needs service and slave read (write) complete interrupts into one interrupt. If a slave read (write) needs service interrupt is active, or queued, and a slave read (write) complete interrupt occurs, and IICx_XTCNTLSS has not yet been read, the two interrupts are merged into a single interrupt. This merge function is performed in the IIC interface logic, and is not under software control.

33.11 General Considerations

1. After a reset, the IIC interface enters the unknown IIC bus state. This state is exited when either activity is seen on the bus or when the exit unknown IIC bus state bit (in the mode control register), is set to a 1. If the IIC interface is being used in a single master system as the master, then the exit unknown IIC bus state bit must be used to force the logic out of the unknown state.
2. Once a byte is written into either the master or slave buffer, a total of four OPB clock periods must occur before the data can be read. Flushing the master or slave buffer also requires four OPB clock periods to complete.
3. IICx_DIRECTCNTL [MSDA,MSC]are used to verify that IICx_DIRECTCNTL [SDAC,SCC] were written successfully, and that the IICSCL signals can be controlled. If IICx_DIRECTCNTL [MSDA, MSC] do not correspond to IICx__DIRECTCNTL [SDAC, SCC], respectively, toggle IICSCL repeatedly to regain control.
4. The master and slave buffers are 4 $\times$ 1 byte-wide FIFOs. Exercise care when using master and slave buffers. As an example, consider the case where one byte of data is written on the IIC bus. The data is first written into the master or slave buffer. After four OPB clock cycles the data is placed on the IIC bus. There is no way to verify data in the buffer without disturbing the IIC transaction. The act of verification requires removing the data. If the data is removed, invalid data is placed on the IIC bus.
5. Use care when monitoring the IICx_XCNTLSS[SDBD] or to IICx_STS[MDBS] to determine when data is present. These bits are set to 1 when the buffer contains data in any stage. Consider the case where the master buffer is empty prior to being loaded with a byte received over the IIC bus. The byte enters the fourth stage of the buffer and the IICx_STS[MDBS] is set to 1. Stages 1, 2, and 3 do not contain data. Therefore, the data is not available for four OPB clock cycles. Any attempt to prematurely read the data yields invalid data.
6. When responding to a slave needs service request, manage the data first. Read data out of the slave buffer, IICx_SDB, for slave reads or write data into the IICx_SDB for slave writes. Next clear the slave needs service request. For reads, clear IICx_XTCNTLSS[SRS]. For writes, clear IICx_XTCNTLSS[SWS]. Last, clear the active interrupt, IICx_STS[IRQA] =0.
7. There is no timeout function implemented in the IIC interface. If this type of error recovery function is needed, it must be implemented in software.

8. Avoid the situations listed in Section 7.2 of the *Phillips Semiconductors I^{2}C Specification,* dated 1995. For your convenience, the section is summarized as follows:

   If multiple masters can be simultaneously involved in a transfer to the same address, or device, then the design of the system must be done in such a way that arbitration between:

   - A repeated Start condition and a data bit does not occur.
   - A Stop condition and a data bit does not occur.
   - A repeated Start condition and a Stop condition does not occur.

   An example of this situation occurs if one master writes 1 byte while another master writes 2 bytes to the same device.

34. GPIO Operations

This chapter describes the General Purpose I/O (GPIO) controllers located on the on-chip peripheral bus (OPB) of the PPC460EX/EXr/GT. The GPIO controllers provide flexible control of multiplexed I/Os selectable under program control. Each of the I/Os is multiplexed with other signals to reduce the quantity of I/O pins needed on the PPC460EX/EXr/GT package.

Note: All shared I/O signals float until configured by setting appropriate bits in the appropriate registers. See *Table 34-7*, *Table 34-8*, and *Table 34-9* for configuration settings.

34.1 Overview

The PPC460EX/EXr/GT has two 32-bit GPIO controllers, GPIO0 and GPIO1. It provides 64 user-programmable external signals, multiplexed with other signal groups.

Configuration registers provide direct control of all GPIO controller functions and I/O signal selections.

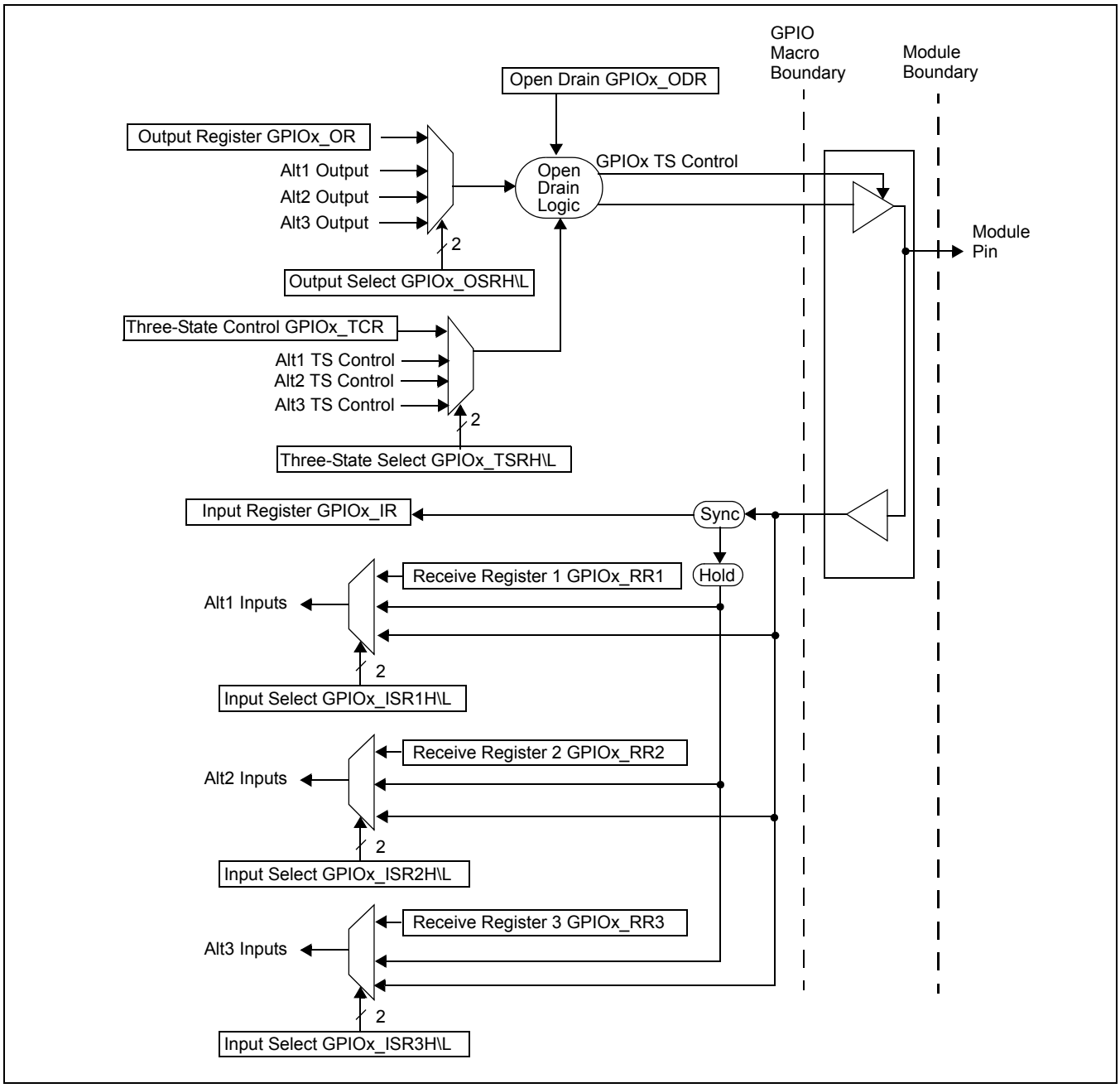
Refer to *Figure 34-1* on page 1234 for an illustration of data flow on the selection and internal function of the GPIO controllers. The module pin serves as both the input and output.

I/O signal selection maintains a bit-for-bit correspondence with input register (GPIO0_IR), output register (GPIO0_OR), three state control register (GPIO0_TCR) and open drain register (GPIO0_ODR). In the PPC460EX/EXr/GT implementation the GPIO0_ODR is only applicable when a GPIO signal is selected. When set, GPIO0_ODR also overrides GPIO0_TCR.

Output buffer operation (three-state output, open-drain output) is controlled by the Open Drain Register (GPIOx_ODR) The Three-State Control Register (GPIOx_TCR) either enables the output driver or forces the output to high impedance. The Three State Select register (GPIOx_TSRH\L) selects between the Three-State Control Register (GPIOx_TCR) and Alternate TS controls. The Output Select register (GPIOx_OSL/H) selects between the Output register (GPIOx_OR) and Alternate Output sources.

A signal input on a GPIO pin is captured in the corresponding bit of the GPIOx_IR register, as well as multiplexed to one other internal functional connection which can be further selected by the GPIOx_ISRnH/L registers.

Figure 34-1. GPIO Data Flow and Configuration Registers



34.2 Features

The GPIO Controllers have the following features:

- Direct control of all GPIO controller functions from registers programmed via memory-mapped addresses
  - Each output can be selected from one of four sources.
  - Each output three-state control can be selected from one of four sources.
  - Selection of the input source for the Alternate 1 input
  - Selection of the input source for the Alternate 2 input
  - Selection of the input source for the Alternate 3 input
  - Programmable receive register settings for one of the possible inputs for the Alternate 1 input
  - Programmable receive register settings for one of the possible inputs for the Alternate 2 input
  - Program able receive register settings for one of the possible inputs for the Alternate 3 input
- Control of 32 bidirectional GPIO module pins
  - Each GPIO output can be set from the corresponding Output Register bit.
  - Each GPIO output has programmable three-state control.
  - Each GPIO output can also be programmed to emulate an open drain output.
  - Each GPIO input is observable from the corresponding Input Register bit.

34.3 Clock and Power Management

The GPIO controllers support clock and power management. Unconditional Sleep (Class 1) power management is implemented for each controller, and is enabled by setting the corresponding CPM0_ER[GPIO] bit.

34.4 GPIO Registers

When an I/O is used as GPIO it is controlled by the corresponding bit in the Output Register GPIOx_OR, the Three-State Control Register GPIOx_TCR, the Open Drain Register GPIOx_ODR, and the Input Register GPIOx_IR. Whether an I/O is used as a GPIO or a functional output is selected using the Output Select Register pair GPIOx_OSRH and GPIOx_OSRL. The source of the three-state control is selected using the Three-State Select Register pair GPIOx_TSRH and GPIOx_TSRL. When an alternate input is used, the source of the alternate input is selected using the Input Select Register GPIOx_ISRnL and GPIOx_ISRnH. There are three input select register pairs; one pair for each Alternate input: Alternate 1, Alternate 2, and Alternate 3 functions. The controller also contains three Receive Registers GPIOx_RR one for each set of Alternate inputs. The Receiver Registers are useful for software debug as they can force alternate inputs to a particular state.

Table 34-1. GPIOx Register Summary

Mnemonic (Note 1)	Register	Address (Note 2)	Access	Page
GPIOx_OR	GPIOx Output	0x4 EF60 0n00	R/W	1236
GPIOx_TCR	GPIOx Three-State Control	0x4 EF60 0n04	R/W	1237
GPIOx_OSRL	GPIOx Output Select (Low)	0x4 EF60 0n08	R/W	1237
GPIOx_OSRH	GPIOx Output Select (High)	0x4 EF60 0n0C	R/W	1237
GPIOx_TSRL	GPIOx Three-State Select (Low)	0x4 EF60 0n10	R/W	1238
GPIOx_TSRH	GPIOx Three-State Select (High)	0x4 EF60 0n14	R/W	1238
GPIOx_ODR	GPIOx Open Drain	0x4 EF60 0n18	R/W	1239
GPIOx_IR	GPIOx Input	0x4 EF60 0n1C	R	1239
GPIOx_RR1	GPIOx Receive Register 1	0x4 EF60 0n20	R/W	1240
GPIOx_RR2	GPIOx Receive Register 2	0x4 EF60 0n24	R/W	1240
GPIOx_RR3	GPIOx Receive Register 3	0x4 EF60 0n28	R/W	1240
GPIOx_ISR1L	GPIOx Input Select 1 Low	0x4 EF60 0n30	R/W	1240
GPIOx_ISR1H	GPIOx Input Select 1 High	0x4 EF60 0n34	R/W	1240
GPIOx_ISR2L	GPIOx Input Select Register 2 Low	0x4 EF60 0n38	R/W	1240
GPIOx_ISR2H	GPIOx Input Select Register 2 High	0x4 EF60 0n3C	R/W	1240
GPIOx_ISR3L	GPIOx Input Select Register 3 Low	0x4 EF60 0n40	R/W	1240
GPIOx_ISR3H	GPIOx Input Select Register 3 High	0x4 EF60 0n44	R/W	1240

Notes:
1. x = 0 or 1.
2. n = B or C.
3. All GPIO registers are memory-mapped and accessed using load/store instructions at the register address.
4. All registers are aligned on word boundaries. The input and output select registers are also aligned on doubleword boundaries.

The following sections provide detailed descriptions of the GPIO controller registers. The GPIO controller is attached to the OPB bus. The GPIOx_IR register is read-only; all other registers are both read- and write-accessible.

34.4.1 GPIO Register Reset Values

When a system reset occurs, all register bits in the GPIO controllers, except GPIOx_IR, are reset to 0. All outputs are placed in high impedance. GPIOx_IR is not reset because it is synchronized to the OPB clock and always tracks the state of the I/O pins.

34.4.2 GPIO Output Register (GPIOx_OR)

When a bit in the GPIO controller is used as a GPIO output, the state of the output is controlled by the value in the GPIOx Output Register, GPIOx_OR. Whether the setting of the bit in the GPIOx_OR is visible on the I/O pin is a function of the settings in the GPIOx_TCR, GPIOx_ODR, GPIOx_OSRH/L, and GPIOx_TSRH/L.

Figure 34-2. GPIO Output Register (GPIOx_OR)			
0:31		GPIO register bits 0 Drive a 0 on the GPIO pin 1 Drive a 1 on the GPIO pin	GPIO0_OR is an output source for GPIO00:31 signals. GPIO1_OR is an output source for GPIO32:63 signals.

34.4.3 GPIO Three-State Control Register (GPIOx_TCR)

The GPIOx_TCR register is one source for three-stating a corresponding output. For each bit in the GPIOx_TCR Register to control the corresponding output, the appropriate 2 bits in the GPIO Three-State Select Register High and Low (GPIOx_TSRH, GPIOx_TSRL) must be programmed to 0b00. If the GPIOx_TCR register is selected; setting a bit to 1 in GPIOx_TCR enables the associated output driver. Clearing the bit to 0 forces the corresponding output into high impedance state. When the same bit is also set in GPIOx_ODR, the output emulates an open-drain output, regardless of the bit setting in GPIOx_TCR.

Figure 34-3. GPIO Three-State Control Register (GPIOx_TCR)			
0:31		GPIO register bits 0 GPIO in a high-Z state 1 GPIO driver enabled	GPIO0_TCR, when selected by GPIO0_TSRH/L, controls GPIO00:31 signals. GPIO1_TCR, when selected by GPIO1_TSRH/L, controls GPIO32:63 signals.

34.4.4 GPIO Output Select Registers (GPIOx_OSRH, GPIOx_OSRL)

The GPIOx_OSR register pair (GPIOx_OSRH, GPIOx_OSRL) determines what signal source is sent to the output pin. The 32 bits that control outputs 0–15 are in the GPIOx_OSRL register, and the 32 bits that control outputs 16–31 are GPIOx_OSRH. For each output there can be up to three sources of output value. Two bits in the GPIOx_OSRH/L register pair are needed for each output. This requires a total of 64 bits to control the 32 output signals for each GPIO controller. The 64 bits are made up of two 32-bit registers. *Table 34-2* shows how these 2 bits control the selection of the signal connected to that GPIO Out pin.

Figure 34-4. GPIO Output Select Register Low (GPIOx_OSRL)			
0:1 2:3 ... 30:31		GPIO Output Select 00 GPIOx_OR[0:15] is the source for GPIOM:N signals 01 Alternate 1 output source 10 Alternate 2 output source 11 Alternate 3 output source	GPIOM:N = GPIO00:15 for GPIO0_OSRL and GPIO0_OR[0:15] GPIOM:N = GPIO32:47 for GPIO1_OSRL and GPIO1_OR[0:15]

Figure 34-5. GPIO Output Select Register High (GPIOx_OSRH)			
0:1 2:3 ... 30:31		GPIO Output Select 00 GPIOx_OR[16:31] is the source for GPIOP:Q signals 00 01 Alternate 1 output source 10 Alternate 2 output source 11 Alternate 3 output source	GPIOP:Q = GPIO16:31 for GPIO0_OSRH and GPIO0_OR[16:31] GPIOP:Q = GPIO48:63 for GPIO1_OSRH and GPIO1_OR[16:31]

Table 34-2. GPIO Output Signal Selection

GPIOx_OSRH/L Bits	GPIO_Out Signal Source
00	GPIOx_OR
01	Alt1 output source
10	Alt2 output source
11	Alt3 output source

34.4.5 GPIO Three-State Select Registers (GPIOx_TSRH, GPIOx_TSRL)

The GPIOx_TSR register pair (GPIOx_TSRH, GPIOx_TSRL) selects the three state source for the GPIO output signals. The 32 bits that control outputs 0–15 are in the GPIOx_TSRL register, and the 32 bits that control outputs 16–31 are GPIOx_TSRH. For each output there there are four sources of three-state control. Two bits in the GPIOx_TSRH/L register pair are needed for each output. This requires a total of 64 bits to control the 32 output signals for each GPIO controller. The 64 bits are made up of two 32-bit registers. *Table 34-3* shows how these 2 bits control the selection of the signal connected to that GPIO_Out pin.

Figure 34-6. GPIO Three-State Select Register Low (GPIOx_TSRL)

0:1 2:3 ... 30:31		GPIO Three-State Source Select 00 GPIOx_TCR[0:15] is the source for GPIOM:N signals 01 Alternate 1 three-state source 10 Alternate 2 three-state source 11 Alternate 3 three-state source	For GPIO0_TSRL and GPIO0_TCR[0:15], GPIOM:N = GPIO00:15. For GPIO1_TSRL and GPIO1_TCR[0:15], GPIOM:N = GPIO32:47.

Figure 34-7. GPIO Three-State Select Register High (GPIOx_TSRH)

0:1 2:3 ... 30:31		GPIO Three-State Source Select 00 GPIOx_TCR[16:31] is the source for GPIOP:Q signals 01 Alternate 1 three-state source 10 Alternate 2 three-state source 11 Alternate 3 three-state source	For GPIO0_TSRH and GPIO0_TCR[16:31], GPIOP:Q = GPIO16:31. For GPIO1_TSRH and GPIO1_TCR[16:31], GPIOP:Q = GPIO48:63.

Table 34-3. GPIO Three-State Selection

GPIOx_TSRH/L Bits	GPIO Three-State Control Source
00	GPIOx_TCR
01	Alternate 1 three-state source
10	Alternate 2 three-state source
11	Alternate 3 three-state source

34.4.6 GPIO Open Drain Register (GPIOx_ODR)

The GPIOx_ODR configures the module I/O three-state driver to emulate open drain drivers on a bit-by-bit basis. *Table 34-4* shows the function of the GPIOx_ODR register and its interaction with the three-state control signal and output signal. While the GPIOx_ODR register can control Alternate 1 outputs, as well as alternate three-state control signals, this feature is not used in the PPC460EX/EXr/GT. For this reason the registers listed in the table are used when a bit is used as a GPIO.

Figure 34-8. GPIO Open Drain Register (GPIOx_ODR)			
0:31		GPIO register bits 0 GPIO not configured as an open drain driver 1 GPIO configured as an open drain driver	GPIO0_ODR controls GPIO00:31 signals. GPIO1_ODR controls GPIO32:63 signals.

Table 34-4. GPIOx_ODR Control Settings

GPIOx_ODR Bit	GPIOx_TCR Bit	GPIOx_OR Bit	State of Module Pin
0	0	x	Forced to high impedance
0	1	0	Driving 0
0	1	1	Driving 1
1	x	0	Driving 0
1	x	1	Forced to high impedance

34.4.7 GPIO Input Register (GPIOx_IR)

The state of each bit in the GPIOx_IR Register reflects the corresponding GPIO controller input signal. All input signals are synchronized to OPBClk before being stored in the GPIOx_IR Register. The GPIOx_IR register is read-only and does not change during a read access. To receive valid data as input from a module pin, the three-state output attached to that module pin is first placed in high impedance by setting the corresponding three-state control bit in the GPIOx_TCR register.

Figure 34-9. GPIO Input Register (GPIOx_IR)			
0:31		GPIO register bits 0 Receive a 0 on the GPIO pin 1 Receive a 1 on the GPIO pin	GPIO0_IR is the input for GPIO00:31 signals. GPIO1_IR is the input for GPIO32:63 signals.

34.4.8 GPIO Input Select Registers (GPIOx_ISRnH, GPIOx_ISRnL)

The register pairs GPIOx_ISRnH and GPIOx_ISRnL (where n = 1, 2, or 3) determine what signal source is used as the input signals. The 32 bits that control outputs 0:15 are in the GPIOx_ISRnL register, and the 32 bits that control outputs 16:31 are GPIOx_ISRnH. For each Alternate n input there can be up to three sources of input value. Two bits in the GPIOx_OSRnH/L register pair are needed for each Alternate n input. This requires a total of 64 bits to control the 32 Alternate n input bits. The 64 bits are made up of two 32-bit registers. *Table 34-5* shows how these 2 bits control the selection of the input. For normal operation the only setting used is 01 which selects the pin input.

Figure 34-10. GPIO Input Select Register n Low (GPIOx_ISRnL)

0:1 2:3 ... 30:31		GPIO Input Select 00 GPIOx_RRn bit 0:15 (Used for software debug only) 01 Input from GPIO pin (not synchronized). Configuration used for Alternate 1, 2, or 3 devices 10 Input from GPIO pin (synchronized to OPB Clk and latched) 11 Reserved	GPIO00:15 input sources are selected by GPIO0_ISRnL. GPIO32:47 input sources are selected by GPIO1_ISRnL.

Figure 34-11. GPIO Input Select Register n High (GPIOx_ISRnH)

0:1 2:3 ... 30:31		GPIO Input Select 00 GPIOx_RRn bit 16:31 (Used for software debug only) 01 Input from GPIO pin (not synchronized). Configuration used for Alternate 1, 2, or 3 devices 10 Input from GPIO pin (synchronized to OPB Clk and latched) 11 Reserved	GPIO16:31 input sources are selected by GPIO0_ISRnH. GPIO48:63 input sources are selected by GPIO1_ISRnH.

Table 34-5. GPIO Alternate Input Signal Selection

GPIOx_ISRnH/L Bits	Alternate n Input Signal
00	GPIOx_RRn
01	Pin input (not synchronized)
10	Hold input when selected
11	Reserved

34.4.9 GPIO Receive Register (GPIOx_RRn)

For normal operation the receive register is not used, as noted in later sections on selecting the Alternate signals. They may however be useful during software debug. There are times when a programmer would like to control input values without having to determine how to cause the system to produce the desired value at an input. By programming the desired input bit in GPIOx_RRn for Alternate n inputs and then setting the appropriate two bits, in the associated GPIOx_ISRnH/L register pair to 00, a desired input value can be set. Note that these registers only affect the alternate inputs, not the GPIOx_IR register.

Figure 34-12. GPIO Receive Register n (GPIOx_RRn)			
0:31		Alternate n input (for software debug only) 0 Alternate n input is 0 1 Alternate n input is 1	n = 1 for Alternate 1 n = 2 for Alternate 2 n = 3 for Alternate 3 GPIO0_RRn is the receiver test input for GPIO00:31 signals. GPIO1_RRn is the receiver test input for GPIO32:63 signals.

34.5 GPIO Signal Assignments

This section describes the signal assignments for the PPC460EX/EXr/GT GPIO controller.

Table 34-6 shows the multiplexed signal assignments for the GPIOx controllers.

When a pin is used as a GPIOx signal, the signal at the pin is controlled by the GPIOx_OR, GPIOx_TCR, GPIOx_ODR registers and observed by the GPIOx_IR register. The values in GPIOx registers GPIOx_RRn and GPIOx_ISRnH/L are *don't care*, so the reset default of 0 is usable. Registers GPIOx_OSRH/L and GPIOx_TSRH/L default to all zeros so they are properly set up as GPIOs. When configuring pins for the Alternate 1, 2, and 3 functions, registers GPIOx_OSRL/H, GPIOx_TSRL/H, and GPIOx_ISRnL/H must be modified according the values in the table.

Table 34-6. GPIO Signal Assignments

GPIO Pin	
GPIO00[USB2HD0]	GPIO32[PerPar2][EOT2/TC2][IRQ9]
GPIO01[USB2HD1]	GPIO33[PerPar3][DMAReq3][IRQ4]
GPIO02[USB2HD2]	GPIO34[$\overline{\text{UART0DCD}}$][$\overline{\text{UART1CTS}}$][UART2Tx]
GPIO03[USB2HD3]	GPIO35[$\overline{\text{UART0DSR}}$][$\overline{\text{UART1RTS}}$][UART2Rx]
GPIO04[USB2HD4]	GPIO36[$\overline{\text{UART0CTS}}$][DMAAck3][UART3Rx]
GPIO05[USB2HD5]	GPIO37[$\overline{\text{UART0RTS}}$][EOT3/TC3][UART3Tx]
GPIO06[USB2HD6]	GPIO38[$\overline{\text{UART0DTR}}$][UART1Tx][IRQ5]
GPIO07[USB2HD7]	GPIO39[$\overline{\text{UART0RI}}$][UART1Rx][IRQ6]
GPIO08[USB2DD0]	GPIO40[IRQ3]
GPIO09[USB2DD1]	GPIO41[$\overline{\text{PerCS1}}$][$\overline{\text{NFCE1}}$]
GPIO10[USB2DD2]	GPIO42[$\overline{\text{PerCS2}}$][$\overline{\text{NFCE2}}$]
GPIO11[USB2DD3]	GPIO43[$\overline{\text{PerCS3}}$][$\overline{\text{NFCE3}}$][DMAReq1][IRQ10]
GPIO12[USB2DD4]	GPIO44[$\overline{\text{PerCS4}}$][DMAAck1][IRQ11]
GPIO13[USB2DD5]	GPIO45[$\overline{\text{PerCS5}}$][EOT1/TC1][IRQ12]
GPIO14[USB2DD6]	GPIO46[PerAddr05][DMAReq0][IRQ13]
GPIO15[USB2DD7]	GPIO47[PerAddr06][DMAAck0][IRQ14]
GPIO16[USB2HStop]	GPIO48[PerAddr07][EOT0/TC0][IRQ15]
GPIO17[USB2HNext]	GPIO49[TrcBS0]
GPIO18[USB2HDir]	GPIO50[TrcBS1]
GPIO19[USB2DStop]	GPIO51[TrcBS2]
GPIO20[USB2DNext]	GPIO52[TrcES0]
GPIO21[USB2DDir]	GPIO53[TrcES1]
GPIO22[$\overline{\text{NFRdyBusy}}$]	GPIO54[TrcES2]
GPIO23[$\overline{\text{NFREn}}$]	GPIO55[TrcES3]
GPIO24[$\overline{\text{NFWEn}}$]	GPIO56[TrcES4]
GPIO25[NFCLE]	GPIO57[TrcTS0]
GPIO26[NFALE]	GPIO58[TrcTS1]
GPIO27[IRQ0]	GPIO59[TrcTS2]
GPIO28[IRQ1]	GPIO60[TrcTS3]
GPIO29[IRQ2]	GPIO61[TrcTS4]
GPIO30[PerPar0][DMAReq2][IRQ7]	GPIO62[TrcTS5]
GPIO31[PerPar1][DMAAck2][IRQ8]	GPIO63[TrcTS6]

34.5.1 Programming the GPIO Alternate 1 Bank

When a pin is configured as an Alternate 1 functional pin, registers GPIOx_OSRL/H, GPIOx_TSRL/H, and GPIOx_ISR1L/H must be set as shown in *Table 34-7*. GPIOx_IR is not used and the values in registers GPIOx_OR and GPIOx_RRn are don't care, so the reset default of 0 is usable. The GPIOx_ODR register defaults to all 0s so it is properly set. The values in the table must be maintained and respective bits in GPIOx_ODR must be 0 for Alternate 1 functions when modifying these registers for pins using the GPIO functions. All values for GPIOx_TCR are "don't care."

Table 34-7. Selecting GPIO Alternate 1 Signals

GPIO	Alternate 1	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO0_OSRL		GPIO0_TSRL		GPIO0_ISR1L	
GPIO00	USB2HD0	I/O	0:1	01	0:1	01	0:1	01
GPIO01	USB2HD1	I/O	2:3	01	2:3	01	2:3	01
GPIO02	USB2HD2	I/O	4:5	01	4:5	01	4:5	01
GPIO03	USB2HD3	I/O	6:7	01	6:7	01	6:7	01
GPIO04	USB2HD4	I/O	8:9	01	8:9	01	8:9	01
GPIO05	USB2HD5	I/O	10:11	01	10:11	01	10:11	01
GPIO06	USB2HD6	I/O	12:13	01	12:13	01	12:13	01
GPIO07	USB2HD7	I/O	14:15	01	14:15	01	14:15	01
GPIO08	USB2DD0	I/O	16:17	01	16:17	01	16:17	01
GPIO09	USB2DD1	I/O	18:19	01	18:19	01	18:19	01
GPIO10	USB2DD2	I/O	20:21	01	20:21	01	20:21	01
GPIO11	USB2DD3	I/O	22:23	01	22:23	01	22:23	01
GPIO12	USB2DD4	I/O	24:25	01	24:25	01	24:25	01
GPIO13	USB2DD5	I/O	26:27	01	26:27	01	26:27	01
GPIO14	USB2DD6	I/O	28:29	01	28:29	01	28:29	01
GPIO15	USB2DD7	I/O	30:31	01	30:31	01	30:31	01

Table 34-7. Selecting GPIO Alternate 1 Signals (Continued)

GPIO	Alternate 1	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO0_OSRH		GPIO0_TSRH		GPIO0_ISR1H	
GPIO16	$\overline{\text{USB2HStop}}$	O	0:1	01	0:1	01	0:1	xx
GPIO17	$\overline{\text{USB2HNext}}$	I	2:3	xx	2:3	01	2:3	01
GPIO18	$\overline{\text{USB2HDir}}$	I	4:5	xx	4:5	01	4:5	01
GPIO19	$\overline{\text{USB2DStop}}$	O	6:7	01	6:7	01	6:7	xx
GPIO20	$\overline{\text{USB2DNext}}$	I	8:9	xx	8:9	01	8:9	01
GPIO21	$\overline{\text{USB2DDir}}$	I	10:11	xx	10:11	01	10:11	01
GPIO22			12:13	xx	12:13	xx	12:13	xx
GPIO23			14:15	xx	14:15	xx	14:15	xx
GPIO24			16:17	xx	16:17	xx	16:17	xx
GPIO25			18:19	xx	18:19	xx	18:19	xx
GPIO26			20:21	xx	20:21	xx	20:21	xx
GPIO27	IRQ0	I	22:23	xx	22:23	01	22:23	01
GPIO28	IRQ1	I	24:25	xx	24:25	01	24:25	01
GPIO29	IRQ2	I	26:27	xx	26:27	01	26:27	01
GPIO30	PerPar0	I/O	28:29	01	28:29	01	28:29	01
GPIO31	PerPar1	I/O	30:31	01	30:31	01	30:31	01
			GPIO1_OSRL		GPIO1_TSRL		GPIO1_ISR1L	
GPIO32	PerPar2	I/O	0:1	01	0:1	01	0:1	01
GPIO33	PerPar3	I/O	2:3	01	2:3	01	2:3	01
GPIO34	$\overline{\text{UART0DCD}}$	I	4:5	xx	4:5	01	4:5	01
GPIO35	$\overline{\text{UART0DSR}}$	I	6:7	xx	6:7	01	6:7	01
GPIO36	$\overline{\text{UART0CTS}}$	I	8:9	xx	8:9	01	8:9	01
GPIO37	$\overline{\text{UART0RTS}}$	O	10:11	01	10:11	01	10:11	xx
GPIO38	$\overline{\text{UART0DTR}}$	O	12:13	01	12:13	01	12:13	xx
GPIO39	$\overline{\text{UART0RI}}$	I	14:15	xx	14:15	01	14:15	01
GPIO40	IRQ3	I	16:17	xx	16:17	01	16:17	01
GPIO41	$\overline{\text{PerCS1}}$/$\overline{\text{NFCE1}}$	O	18:19	01	18:19	01	18:19	xx
GPIO42	$\overline{\text{PerCS2}}$/$\overline{\text{NFCE2}}$	O	20:21	01	20:21	01	20:21	xx
GPIO43	$\overline{\text{PerCS3}}$/$\overline{\text{NFCE2}}$	O	22:23	01	22:23	01	22:23	xx
GPIO44	$\overline{\text{PerCS4}}$	O	24:25	01	24:25	01	24:25	xx
GPIO45	$\overline{\text{PerCS5}}$	O	26:27	01	26:27	01	26:27	xx
GPIO46	PerAddr05	O	28:29	01	28:29	01	28:29	xx
GPIO47	PerAddr06	O	30:31	01	30:31	01	30:31	xx

Table 34-7. Selecting GPIO Alternate 1 Signals (Continued)

GPIO	Alternate 1	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO1_OSRH		GPIO1_TSRH		GPIO1_ISR1H	
GPIO48	PerAddr07	O	0:1	01	0:1	01	0:1	xx
GPIO49	TrcBS0	O	2:3	01	2:3	01	2:3	xx
GPIO50	TrcBS1	O	4:5	01	4:5	01	4:5	xx
GPIO51	TrcBS2	O	6:7	01	6:7	01	6:7	xx
GPIO52	TrcES0	O	8:9	01	8:9	01	8:9	xx
GPIO53	TrcES1	O	10:11	01	10:11	01	10:11	xx
GPIO54	TrcES2	O	12:13	01	12:13	01	12:13	xx
GPIO55	TrcES3	O	14:15	01	14:15	01	14:15	xx
GPIO56	TrcES4	O	16:17	01	16:17	01	16:17	xx
GPIO57	TrcTS0	O	18:19	01	18:19	01	18:19	xx
GPIO58	TrcTS1	O	20:21	01	20:21	01	20:21	xx
GPIO593	TrcTS2	O	22:23	01	22:23	01	22:23	xx
GPIO60	TrcTS3	O	24:25	01	24:25	01	24:25	xx
GPIO61	TrcTS4	O	26:27	01	26:27	01	26:27	xx
GPIO62	TrcTS5	O	28:29	01	28:29	01	28:29	xx
GPIO63	TrcTS6	O	30:31	01	30:31	01	30:31	xx

Notes:
1. Programming a GPIO_TCR single bit to 0 and GPIO_TSRL/H double bits to 00 are also valid for all inputs being only GPIO.
2. Programming a GPIO_TCR single bit to 1 and GPIO_TSRL/H double bits to 00 are also valid for all outputs being only GPIO.

34.5.2 Programming the GPIO Alternate 2 Bank

When a pin is configured as an Alternate 2 functional pin, registers GPIOx_OSRL/H, GPIOx_TSRL/H, and GPIOx_ISR2L/H must be set as shown in *Table 34-8*. GPIOx_IR is not used and the values in registers GPIOx_OR and GPIOx_RRn are don't care, so the reset default of 0 is usable. The GPIOx_ODR register defaults to all 0s so it is properly set. The values in the table must be maintained and respective bits in GPIOx_ODR must be 0 for Alternate 2 functions when modifying these registers for pins using the GPIO functions. All values for GPIOx_TCR are "don't care."

Table 34-8. Selecting GPIO Alternate 2 Signals

GPIO	Alternate 2	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO0_OSRL		GPIO0_TSRL		GPIO0_ISR2L	
GPIO00			0:1	xx	0:1	xx	0:1	xx
GPIO01			2:3	xx	2:3	xx	2:3	xx
GPIO02			4:5	xx	4:5	xx	4:5	xx
GPIO03			6:7	xx	6:7	xx	6:7	xx
GPIO04			8:9	xx	8:9	xx	8:9	xx
GPIO05			10:11	xx	10:11	xx	10:11	xx
GPIO06			12:13	xx	12:13	xx	12:13	xx
GPIO07			14:15	xx	14:15	xx	14:15	xx
GPIO08			16:17	xx	16:17	xx	16:17	xx
GPIO09			18:19	xx	18:19	xx	18:19	xx
GPIO10			20:21	xx	20:21	xx	20:21	xx
GPIO11			22:23	xx	22:23	xx	22:23	xx
GPIO12			24:25	xx	24:25	xx	24:25	xx
GPIO13			26:27	xx	26:27	xx	26:27	xx
GPIO14			28:29	xx	28:29	xx	28:29	xx
GPIO15			30:31	xx	30:31	xx	30:31	xx

Table 34-8. Selecting GPIO Alternate 2 Signals (Continued)

GPIO	Alternate 2	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO0_OSRH		GPIO0_TSRH		GPIO0_ISR2H	
GPIO16			0:1	xx	0:1	xx	0:1	xx
GPIO17			2:3	xx	2:3	xx	2:3	xx
GPIO18			4:5	xx	4:5	xx	4:5	xx
GPIO19			6:7	xx	6:7	xx	6:7	xx
GPIO20			8:9	xx	8:9	xx	8:9	xx
GPIO21			10:11	xx	10:11	xx	10:11	xx
GPIO22			12:13	xx	12:13	xx	12:13	xx
GPIO23			14:15	xx	14:15	xx	14:15	xx
GPIO24			16:17	xx	16:17	xx	16:17	xx
GPIO25			18:19	xx	18:19	xx	18:19	xx
GPIO26			20:21	xx	20:21	xx	20:21	xx
GPIO27			22:23	xx	22:23	xx	22:23	xx
GPIO28			24:25	xx	24:25	xx	24:25	xx
GPIO29			26:27	xx	26:27	xx	26:27	xx
GPIO30	DMAReq2	I	28:29	xx	28:29	10	28:29	01
GPIO31	DMAAck2	O	30:31	10	30:31	10	30:31	xx
			GPIO1_OSRL		GPIO1_TSRL		GPIO1_ISR2L	
GPIO32	EOT2/TC2	I/O	0:1	10	0:1	10	0:1	01
GPIO33	DMAReq3	I	2:3	xx	2:3	10	2:3	01
GPIO34	$\overline{\text{UART1CTS}}$	I	4:5	xx	4:5	10	4:5	01
GPIO35	$\overline{\text{UART1RTS}}$	O	6:7	10	6:7	10	6:7	xx
GPIO36	DMAAck3	O	8:9	10	8:9	10	8:9	xx
GPIO37	EOT3/TC3	I/O	10:11	10	10:11	10	10:11	01
GPIO38	UART1Tx	O	12:13	10	12:13	10	12:13	xx
GPIO39	UART1Rx	I	14:15	xx	14:15	10	14:15	01
GPIO40			16:17	xx	16:17	xx	16:17	xx
GPIO41			18:19	xx	18:19	xx	18:19	xx
GPIO42			20:21	xx	20:21	xx	20:21	xx
GPIO43	DMAReq1	I	22:23	xx	22:23	10	22:23	01
GPIO44	DMAAck1	O	24:25	10	24:25	10	24:25	xx
GPIO45	EOT1/TC1	I/O	26:27	10	26:27	10	26:27	01
GPIO46	DMAReq0	I	28:29	xx	28:29	10	28:29	01
GPIO47	DMAAck0	O	30:31	10	30:31	10	30:31	xx

Table 34-8. Selecting GPIO Alternate 2 Signals (Continued)

GPIO	Alternate 2	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO1_OSRH		GPIO1_TSRH		GPIO1_ISR2H	
GPIO48	EOT0/TC0	I/O	0:1	10	0:1	10	0:1	01
GPIO49			2:3	xx	2:3	xx	2:3	xx
GPIO50			4:5	xx	4:5	xx	4:5	xx
GPIO51			6:7	xx	6:7	xx	6:7	xx
GPIO52			8:9	xx	8:9	xx	8:9	xx
GPIO53			10:11	xx	10:11	xx	10:11	xx
GPIO54			12:13	xx	12:13	xx	12:13	xx
GPIO55			14:15	xx	14:15	xx	14:15	xx
GPIO56			16:17	xx	16:17	xx	16:17	xx
GPIO57			18:19	xx	18:19	xx	18:19	xx
GPIO58			20:21	xx	20:21	xx	20:21	xx
GPIO593			22:23	xx	22:23	xx	22:23	xx
GPIO60			24:25	xx	24:25	xx	24:25	xx
GPIO61			26:27	xx	26:27	xx	26:27	xx
GPIO62			28:29	xx	28:29	xx	28:29	xx
GPIO63			30:31	xx	30:31	xx	30:31	xx

Notes:
1. Programming a GPIO_TCR single bit to 0 and GPIO_TSRL/H double bits to 00 are also valid for all inputs being only GPIO.
2. Programming a GPIO_TCR single bit to 1 and GPIO_TSRL/H double bits to 00 are also valid for all outputs being only GPIO.

34.5.3 Programming the GPIO Alternate 3 Bank

When a pin is configured as an Alternate 3 functional pin, registers GPIOx_OSRL/H, GPIOx_TSRL/H, and GPIOx_ISR3L/H must be set as shown in *Table 34-9*. GPIOx_IR is not used and the values in registers GPIOx_OR and GPIOx_RRn are "don't care", so the reset default of 0 is usable. The GPIOx_ODR register defaults to all 0s so it is properly set. The values in the table must be maintained and respective bits in GPIOx_ODR must be 0 for Alternate 3 functions when modifying these registers for pins using the GPIO functions. All values for GPIOx_TCR are "don't care."

Table 34-9. Selecting GPIO Alternate 3 Signals

GPIO	Alternate 3	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO0_OSRL		GPIO0_TSRL		GPIO0_ISR3L	
GPIO00			0:1	xx	0:1	xx	0:1	xx
GPIO01			2:3	xx	2:3	xx	2:3	xx
GPIO02			4:5	xx	4:5	xx	4:5	xx
GPIO03			6:7	xx	6:7	xx	6:7	xx
GPIO04			8:9	xx	8:9	xx	8:9	xx
GPIO05			10:11	xx	10:11	xx	10:11	xx
GPIO06			12:13	xx	12:13	xx	12:13	xx
GPIO07			14:15	xx	14:15	xx	14:15	xx
GPIO08			16:17	xx	16:17	xx	16:17	xx
GPIO09			18:19	xx	18:19	xx	18:19	xx
GPIO10			20:21	xx	20:21	xx	20:21	xx
GPIO11			22:23	xx	22:23	xx	22:23	xx
GPIO12			24:25	xx	24:25	xx	24:25	xx
GPIO13			26:27	xx	26:27	xx	26:27	xx
GPIO14			28:29	xx	28:29	xx	28:29	xx
GPIO15			30:31	xx	30:31	xx	30:31	xx

Table 34-9. Selecting GPIO Alternate 3 Signals (Continued)

GPIO	Alternate 3	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO0_OSRH		GPIO0_TSRH		GPIO0_ISR3H	
GPIO16			0:1	xx	0:1		0:1	xx
GPIO17			2:3	xx	2:3	xx	2:3	xx
GPIO18			4:5	xx	4:5	xx	4:5	xx
GPIO19			6:7	xx	6:7	xx	6:7	xx
GPIO20			8:9	xx	8:9	xx	8:9	xx
GPIO21			10:11	xx	10:11	xx	10:11	xx
GPIO22			12:13	xx	12:13	xx	12:13	xx
GPIO23			14:15	xx	14:15	xx	14:15	xx
GPIO24			16:17	xx	16:17	xx	16:17	xx
GPIO25			18:19	xx	18:19	xx	18:19	xx
GPIO26			20:21	xx	20:21	xx	20:21	xx
GPIO27			22:23	xx	22:23	xx	22:23	xx
GPIO28			24:25	xx	24:25	xx	24:25	xx
GPIO29			26:27	xx	26:27	xx	26:27	xx
GPIO30	IRQ7	I	28:29	xx	28:29	11	28:29	01
GPIO31	IRQ8	I	30:31	xx	30:31	11	30:31	01
			GPIO1_OSRL		GPIO1_TSRL		GPIO1_ISR3L	
GPIO32	IRQ9	I	0:1	xx	0:1	11	0:1	01
GPIO33	IRQ4	I	2:3	xx	2:3	11	2:3	01
GPIO34	UART2TX	O	4:5	11	4:5	11	4:5	xx
GPIO35	UART2RX	I	6:7	xx	6:7	11	6:7	01
GPIO36	UART3RX	I	8:9	xx	8:9	11	8:9	01
GPIO37	UART3TX	O	10:11	11	10:11	11	10:11	xx
GPIO38	IRQ5	I	12:13	xx	12:13	11	12:13	01
GPIO39	IRQ6	I	14:15	xx	14:15	11	14:15	01
GPIO40			16:17	xx	16:17	xx	16:17	xx
GPIO41			18:19	xx	18:19	xx	18:19	xx
GPIO42			20:21	xx	20:21	xx	20:21	xx
GPIO43	IRQ10	I	22:23	xx	22:23	11	22:23	01
GPIO44	IRQ11	I	24:25	xx	24:25	11	24:25	01
GPIO45	IRQ12	I	26:27	xx	26:27	11	26:27	01
GPIO46	IRQ13	I	28:29	xx	28:29	11	28:29	01
GPIO47	IRQ14	I	30:31	xx	30:31	11	30:31	01

Table 34-9. Selecting GPIO Alternate 3 Signals (Continued)

GPIO	Alternate 3	I/O	Bits	Value	Bits	Value	Bits	Value
			GPIO1_OSRH		GPIO1_TSRH		GPIO1_ISR3H	
GPIO48	IRQ15	I	0:1	xx	0:1	11	0:1	01
GPIO49			2:3	xx	2:3	xx	2:3	xx
GPIO50			4:5	xx	4:5	xx	4:5	xx
GPIO51			6:7	xx	6:7	xx	6:7	xx
GPIO52			8:9	xx	8:9	xx	8:9	xx
GPIO53			10:11	xx	10:11	xx	10:11	xx
GPIO54			12:13	xx	12:13	xx	12:13	xx
GPIO55			14:15	xx	14:15	xx	14:15	xx
GPIO56			16:17	xx	16:17	xx	16:17	xx
GPIO57			18:19	xx	18:19	xx	18:19	xx
GPIO58			20:21	xx	20:21	xx	20:21	xx
GPIO593			22:23	xx	22:23	xx	22:23	xx
GPIO60			24:25	xx	24:25	xx	24:25	xx
GPIO61			26:27	xx	26:27	xx	26:27	xx
GPIO62			28:29	xx	28:29	xx	28:29	xx
GPIO63			30:31	xx	30:31	xx	30:31	xx

Notes:
1. Programming a GPIO_TCR single bit to 0 and GPIO_TSRL/H double bits to 00 are also valid for all inputs being only GPIO.
2. Programming a GPIO_TCR single bit to 1 and GPIO_TSRL/H double bits to 00 are also valid for all outputs being only GPIO.

35. General Purpose Timer

The General Purpose Timer (GPT) is a system timer with seven maskable compare registers and a 32-bit time base counter. Each compare register has a corresponding GPT interrupt to the UIC. GPT interrupts can be generated for a specific count by a match between the contents of a compare register and the time base counter. GPT interrupts may also be generated on a specific interval by masking individual bits of a compare register. The following sections provide an overview, programming steps and register descriptions.

35.1 GPT Features

- 32-bit time base counter driven by the OPB bus clock
- OPB slave interface for access to all control, timer and status registers which provide direct control of all GPT functions
- Seven compare timers and corresponding interrupt outputs
- Programmable time base register (sets the time base counter)
- Maskable time-base comparison support for each compare timer
- Programmable compare timer values
- Enable/disable control of all compare interrupts
- Mask control of interrupt status bits

35.2 Time Base Counter

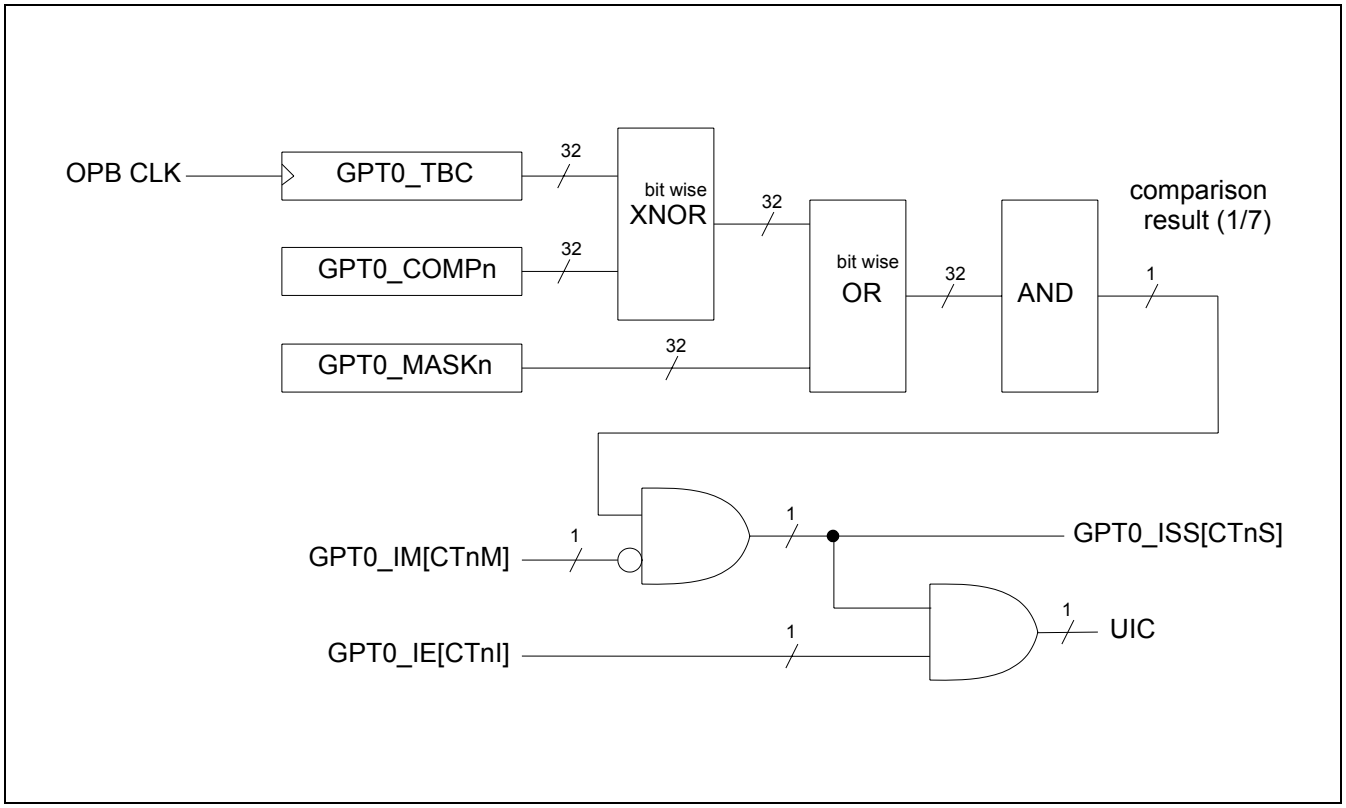
The Time Base Counter (TBC) is both an OPB register and an unsigned counter, and provides the reference time for all compare timers. It increments by one with each clock period and is 32-bit wide. When the time base counter is at its maximum value (all bits set to 1) it rolls back to zero upon the next clock.

The TBC is synchronously reset to zero upon a full chip reset. It may be read and written via software through the OPB interface. When written the new value is stored with the next rising edge of OPBClk.

35.3 Compare Timers

The time base counter, GPT0_TBC, is incremented each OPB clock period and evaluated for equivalence to the compare registers as illustrated in *Figure 35-1*. The XNOR identifies bits in the time base counter that are equivalent to corresponding bits in a compare register GTP0_COMP0:6. The 32-bit output of the XNOR is OR'ed with a 32-bit compare mask GPT0_MASK0:6. Bits set to 1 in the compare mask are masked and are not evaluated for equivalence. The comparison result of the 32-bit input AND reduces the masked result of the OR to a single bit indicating equivalence if set to 1. The status register GPT0_ISS records the comparison if the corresponding compare result is unmasked by the interrupt mask register field GPT0_IM[CTnM] = 0. To generate a GPT interrupt the interrupt enable bit GPT0_IE[CTnI] = 1 must also be enabled. *Figure 35-1* illustrates the comparison of the time base counter to a compare register.

Figure 35-1. Time Base Counter and Compare Register



35.3.1 Compare Timer Interrupt

The following are steps for enabling a GPT interrupt:

1. Set the corresponding compare register (GPT0_COMPn) to the desired compare value.
2. Set the corresponding compare mask register (GPT0_MASKn) with the desired mask bit pattern.
3. Unmask the GPT interrupt by clearing the interrupt mask bit (GPT0_IM[CTnM]=0).
4. Enable the GPT interrupt by setting the enable bit (GPT0_IE[CTnI]=1).
5. Configure the UIC to enable a GPT interrupt (see *Interrupt Controller Operations* on page 255).

Note: Seven separate interrupt lines (UIC 12:18), one for each of the seven compare timers, are implemented.

35.4 GPT Registers

All GPT registers are memory mapped and accessed via load/store instructions at the address of the register. The registers are accessed from the OPB on 32-bit boundaries relative to the configurable base address, 0x4 EF60 0000. The GPT Interrupt Status Register (GPT0_IS) bits are either set or cleared when written, depending upon which one of two addresses are used. All other registers are both read and write accessible in the normal manner.

Table 35-1. GPT Registers

Mnemonic	Register	Address	Access	Page
GPT0_TBC	Time Base Counter	0x4 EF60 0000	R/W	1255
GPT0_IM	GPT Interrupt Mask	0x4 EF60 0018	R/W	1256
GPT0_ISS	GPT Interrupt Status (Set bits if write 1)	0x4 EF60 001C	R/W	1256
GPT0_ISC	GPT Interrupt Status (Clear bits if write 1)	0x4 EF60 0020	R/W	1256
GPT0_IE	GPT Interrupt Enable	0x4 EF60 0024	R/W	1257
GPT0_COMP0	Compare Timer 0	0x4 EF60 0080	R/W	1258
GPT0_COMP1	Compare Timer 1	0x4 EF60 0084	R/W	1258
GPT0_COMP2	Compare Timer 2	0x4 EF60 0088	R/W	1258
GPT0_COMP3	Compare Timer 3	0x4 EF60 008C	R/W	1258
GPT0_COMP4	Compare Timer 4	0x4 EF60 0090	R/W	1258
GPT0_COMP5	Compare Timer 5	0x4 EF60 0094	R/W	1258
GPT0_COMP6	Compare Timer 6	0x4 EF60 0098	R/W	1258
GPT0_MASK0	Compare Mask (Compare Timer 0)	0x4 EF60 00C0	R/W	1258
GPT0_MASK1	Compare Mask (Compare Timer 1)	0x4 EF60 00C4	R/W	1258
GPT0_MASK2	Compare Mask (Compare Timer 2)	0x4 EF60 00C8	R/W	1258
GPT0_MASK3	Compare Mask (Compare Timer 3)	0x4 EF60 00CC	R/W	1258
GPT0_MASK4	Compare Mask (Compare Timer 4)	0x4 EF60 00D0	R/W	1258
GPT0_MASK5	Compare Mask (Compare Timer 5)	0x4 EF60 00D4	R/W	1258
GPT0_MASK6	Compare Mask (Compare Timer 6)	0x4 EF60 00D8	R/W	1258
GPT0_DCT0	Down Count Timer	0x4 EF60 0110	R/W	1258
GPT0_DCIS	Down Count Timer Interrupt Status	0x4 EF60 011C	R/W	1259

35.4.1 GPT Time Base Counter Register (GPT0_TBC)

GPT time base counter register (GPT0_TBC) is used by the compare timers as a reference for determining event occurrences and for software to use as a general timer.

Figure 35-2. GPT Time Base Counter Register (GPT0_TBC)

0:31	TB	Time Base

35.4.2 GPT Interrupt Mask Register (GPT0_IM)

GPT interrupt mask register (GPT0_IM) bits correspond to the compare timer interrupt masks. The register bits mask both the setting of the corresponding GPT0_IS bits and the interrupt output signals to the UIC. If masked, GPT0_IS bits are not set, and interrupt signals are not generated (even if the GPT0_IE bits are enabled). For interrupt signals to be active, the GPT0_IM bits must be reset (not masked) and the GPT0_IE bits must be enabled.

Figure 35-3. GPT Interrupt Mask Register (GPT0_IM)

0:15		Reserved	
16	CT0M	Compare Timer 0 Interrupt Mask 0 Compare timer 0 interrupt mask disabled 1 Compare timer 0 interrupt mask enabled	
17	CT1M	Compare Timer 1 Interrupt Mask 0 Compare timer 1 interrupt mask disabled 1 Compare timer 1 interrupt mask enabled	
18	CT2M	Compare Timer 2 Interrupt Mask 0 Compare timer 2 interrupt mask disabled 1 Compare timer 2 interrupt mask enabled	
19	CT3M	Compare Timer 3 Interrupt Mask 0 Compare timer 3 interrupt mask disabled 1 Compare timer 3 interrupt mask enabled	
20	CT4M	Compare Timer 4 Interrupt Mask 0 Compare timer 4 interrupt mask disabled 1 Compare timer 4 interrupt mask enabled	
21	CT5M	Compare Timer 5 Interrupt Mask 0 Compare timer 5 interrupt mask disabled 1 Compare timer 5 interrupt mask enabled	
22	CT6M	Compare Timer 6 Interrupt Mask 0 Compare timer 6 interrupt mask disabled 1 Compare timer 6 interrupt mask enabled	
23:31		Reserved	

35.4.3 GPT Interrupt Status Register (GPT0_ISS and GPT0_ISC)

GPT interrupt status register (GPT0_ISS/ICC) bits correspond to the compare timer interrupt status. The GPT Interrupt status bits for the compare timers are set when a valid comparison is made, and the compare interrupt is enabled.

GPT0_ISS can be accessed through address offset 0x1C, which provides a normal read access and a "Write-Set" access, allowing individual status bits to be set through a write access. Any status bits written to 1 are set (forced to 1), while bits written to 0 remain unchanged (0 or 1).

Offset 0x20 (GPT0_ISC) provides a normal read access and a "Write-Clear" access, which allows individual status bits to be reset through a write access. Any status bits written to 1 are cleared (forced to 0), while bits written to 0 remain unchanged (0 or 1).

Figure 35-4. GPT Interrupt Status Register (GPT0_ISS and GPT0_ISC)			
0:15		Reserved	
16	CT0S	Compare Timer 0 Interrupt Status 0 Compare timer 0 interrupt status disabled 1 Compare timer 0 interrupt status enabled	
17	CT1S	Compare Timer 1 Interrupt Status 0 Compare timer 1 interrupt status disabled 1 Compare timer 1 interrupt status enabled	
18	CT2IS	Compare Timer 2 Interrupt Status 0 Compare timer 2 interrupt status disabled 1 Compare timer 2 interrupt status enabled	
19	CT3S	Compare Timer 3 Interrupt Status 0 Compare timer 3 interrupt status disabled 1 Compare timer 3 interrupt status enabled	
20	CT4S	Compare Timer 4 Interrupt Status 0 Compare timer 4 interrupt status disabled 1 Compare timer 4 interrupt status enabled	
21	CT5S	Compare Timer 5 Interrupt Status 0 Compare timer 5 interrupt status disabled 1 Compare timer 5 interrupt status enabled	
22	CT6S	Compare Timer 6 Interrupt Status 0 Compare timer 6 interrupt status disabled 1 Compare timer 6 interrupt status enabled	
23:31		Reserved	

35.4.4 GPT Interrupt Enable Register (GPT0_IE)

GPT interrupt enable register (GPT0_IE) bits correspond to the compare timer interrupt enable bits. When set, GPT0_IE bits prevent the corresponding compare interrupts from activating the GPT UIC interrupts, even if the interrupt mask (GPT0_IM) bits are set; however, these bits have no effect on the corresponding interrupt status (GPT0_IS) bits.

Figure 35-5. GPT Interrupt Enable Register (GPT0_IE)			
0:15		Reserved	
16	CT0I	Compare Timer 0 Interrupt Enable 0 Compare timer 0 interrupt enable disabled 1 Compare timer 0 interrupt enable enabled	
17	CT1I	Compare Timer 1 Interrupt Enable 0 Compare timer 1 interrupt enable disabled 1 Compare timer 1 interrupt enable enabled	
18	CT2I	Compare Timer 2 Interrupt Enable 0 Compare timer 2 interrupt enable disabled 1 Compare timer 2 interrupt enable enabled	
19	CT3I	Compare Timer 3 Interrupt Enable 0 Compare timer 3 interrupt enable disabled 1 Compare timer 3 interrupt enable enabled	

20	CT4I	Compare Timer 4 Interrupt Enable 0 Compare timer 4 interrupt enable disabled 1 Compare timer 4 interrupt enable enabled	
21	CT5I	Compare Timer 5 Interrupt Enable 0 Compare timer 5 interrupt enable disabled 1 Compare timer 5 interrupt enable enabled	
22	CT6I	Compare Timer 6 Interrupt Enable 0 Compare timer 6 interrupt enable disabled 1 Compare timer 6 interrupt enable enabled	
23:31		Reserved	

35.4.5 GPT Compare Timer Registers (GPT0_COMP0–GPT0_COMP6)

Each GPT compare timer register (GPT0_COMP0:GPT0_COMP6) is programmed with the value that is continually compared to the TBC value, as filtered through each MASK register. The width of each GPT0_COMP0:GPT0_COMP6 is 32 bits.

Figure 35-6. GPT Compare Timer Register (GPT0_COMP0–GPT0_COMP6)

0:31	COMP	Compare Timer	

35.4.6 GPT Compare Mask Registers (GPT0_MASK0–GPT0_MASK6)

GPT compare mask registers (GPT0_MASK0:GPT0_MASK6) bits are used by the compare timers to mask off the comparison (i.e., force a valid compare) of individual bits when the comparison function is performed. For bits that are set, a valid compare is always assumed, regardless of the actual value of these bits in the GPT0_COMP0:GPT0_COMP6 or GPT0_TBC registers. The width of each implemented Mask Register is 32 bits.

Figure 35-7. GPT Compare Mask Register (GPT0_MASK0–GPT0_MASK6)

0:31	MASK	Comparison Function 0 Comparison enabled 1 Comparison disabled	When set to 1, a valid comparison is assumed.

35.4.7 GPT Down Count Timer (GPT0_DCT0)

The GPT0_DCT0 is loaded with the time or count value to be loaded into the down counter. It is a full 32-bit quantity that is decremented each TBCClk cycle until the counter reaches zero, where it stops until another value is loaded. If the register is loaded with 0xFFFFFFFF, the counter will not count.

Figure 35-8. Down Count Timer Register (GPT0_DCT0)

0:31	DCT	Time value

35.4.8 GPT Down Count Timer Interrupt Status (GPT0_DCIS)

The GPT0_DCIS is used to determine that the down count timer caused the interrupt.

Figure 35-9. Down Count Timer Register (GPT0_DCIS)

0	DCIS	Down Count Interrupt Status 0 Counter not expired 1 Counter expired	Writing a 1 resets the bit.
1:31		Reserved	

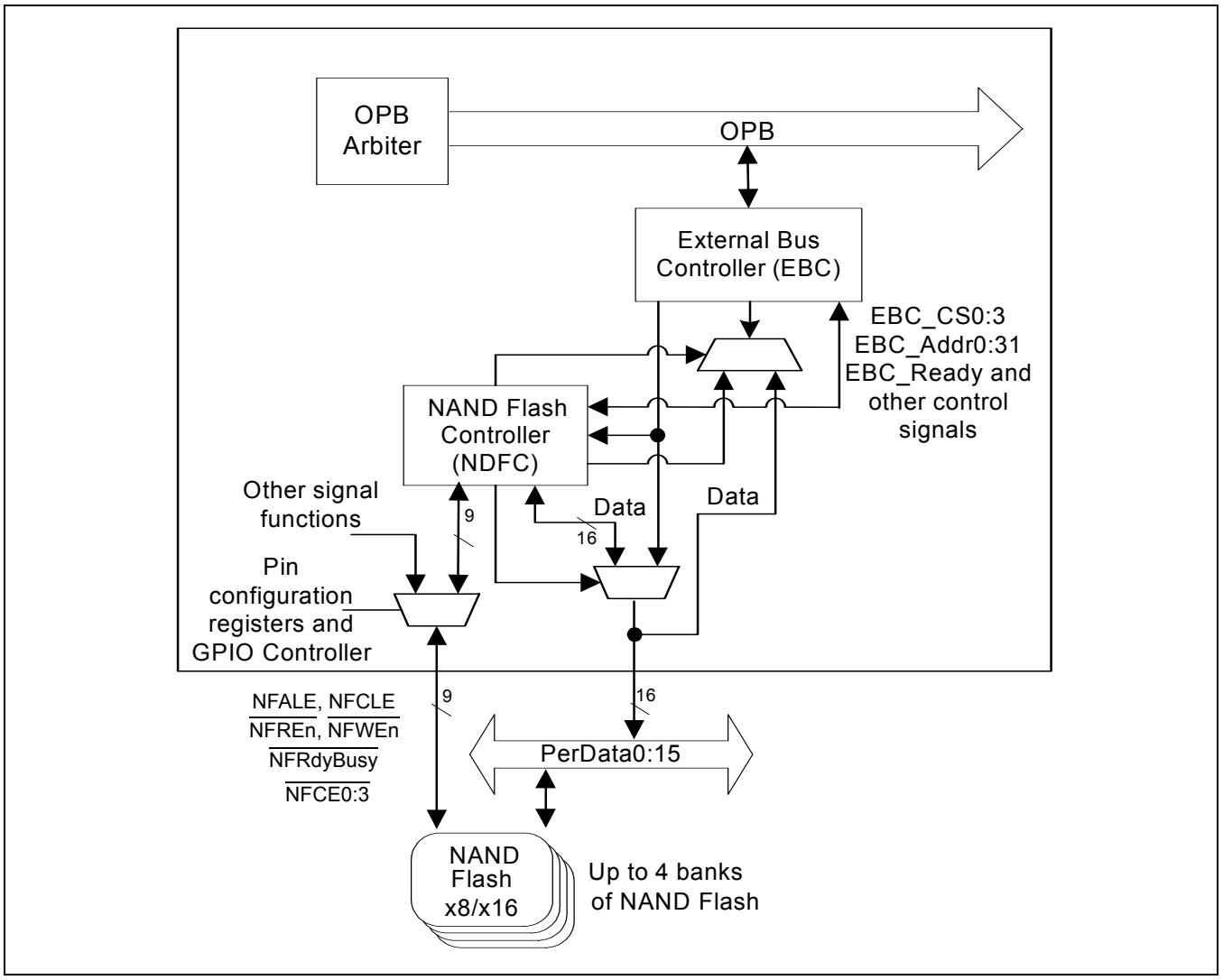
36. NAND Flash Controller

The NAND Flash controller (NDFC) enables glueless interfacing to NAND Flash.

NDFC has the following features:

- Direct interfacing to discrete NAND Flash devices (up to 4 devices)
- Device Sizes of 4MB and larger supported for read/write access
- ECC generation—hamming code, single bit correction, double bit detection (SEC/DED)
- x8 wide command write, x8 wide address write, x8 and x16 bit wide data read/write
- Automatically generates RE# and WE# strobes with configurable strobe pulse width parameter
- Supports DMA in memory-to-memory copying mode to allow direct, no-processor-intervention block copy from NAND Flash out to DDR (after initial command/address setup performed)
- Interrupt on device becoming ready (after long page write or block erase operations).
- Boot-from-NAND: Execute a linear sequence of boot code out of the first 4KB of Block 0.
- Operates from EBC peripheral clock.
- Frequency range for operation: Up to 100MHz
- Class 2 power management

Figure 36-1. Typical System Application



36.1 Overview

The NDFC supports up to four NAND Flash devices. It provides direct command, address, and data access to the external device as well as a 4KB memory mapped linear access region. The 4KB linear access region enables boot code to execute up to 4KB directly from NAND Flash. NDFC also provides hardware support for the Hamming code ECC algorithm that assist software in 1-bit error correction and 2-bit error detection (per 256B of data).

36.1.1 Supported NAND Flash Size

There are no limits on the NAND flash size. NAND flash does not occupy the memory map because it is accessed through the NAND flash controller registers. Software addresses NAND flash by writing commands and addresses to the NDFC0_CMD and NDFC0_ADDR registers. Data is accessed a byte or half-word at a time by reading or writing the NDCF0_DATA register as described in *Reading NAND Flash* on page 1270 and *Writing NAND Flash* on page 1271.

When booting directly from NAND flash, there are some limitations due to the page size. The NAND flash must have 512 byte or 2KB pages in order for the full 4KB linear region to be accessible using the auto-read mode as described in *AutoRead Mode* on page 1268. Booting from NAND flash with 4KB pages is possible but the linear region is limited to 2KB. *Booting from NDFC* on page 1266 describes the NAND flash boot process.

36.1.2 SLC and MLC NAND Flash Support

The NAND flash controller supports either SLC (single-level cell) or MLC (multi-level cell) NAND. MLC NAND, however, should only be accessed through a file system that manages wear-leveling, error detection and error correction in software. SLC NAND is less susceptible to soft-errors and can be accessed without a file system but still requires error detection and error correction in software.

36.1.3 ECC Generation/Detection

The NAND flash controller provides ECC generation in hardware for page accesses as described in *ECC* on page 1271. The hardware assisted ECC code generation is intended for SLC NAND. The Hamming code ECC algorithm is not adequate for MLC NAND.

Errors are detected on a page read by software using the ECC value accumulated in the NDFC0_ECC0:7 registers. The accumulated value is compared with the value read from NAND flash extended/spare area. When the two ECC codes do not match, there is an error. The type of error depend on the ECC syndrome code results described in *Table 36-7* on page 1272.

36.2 NDFC Signal Multiplexing

NDFC shares I/O with EBC and GPIO signals. In order to select the NDFC signals, SDR0_CUST0[MEN] must be set to 0x2. Additionally, SDR0_CUST0[NGC] must be configure to select the NDFC chip enables (NFCE0:3) as these signals are shared with the EBC (PerCS0:3). The NDFC data signals are shared dynamically with EBC (PerData00:15) depending on the access.

36.3 NDFC Interface

The NDFC interface has standard NAND Flash control signals which can be directly connected to NAND Flash. The NDFC data interface however uses the external EBC interface signals PerData0:15. These signals are numbered using IBM bit numbering. PerData0 is the msb of the LSB. PerData8 is the msb of the MSB.

For 8- and 16-bit NAND flash, connect PerData0:7 to NAND Flash data 7:0. Commands are driven on PerData0:7 for both interface widths.

For an 8-bit NAND Flash, connect as follows:

PerData0 to NAND Flash data 7
PerData1 to NAND Flash data 6
.
.
.
PerData7 to NAND Flash data 0

For a 16-bit NAND Flash, connect as follows:

PerData0 to NAND Flash data 7
PerData1 to NAND Flash data 6
.
.
.
PerData7 to NAND Flash data 0

PerData15 to NAND Flash data 8
PerData14 to NAND Flash data 9
.
.
.
PerData8 to NAND Flash data 15

36.4 Resetting the Controller

Software should use SDR0_SRST1[NDFC] to reset the controller.

36.5 Configuring EBC to Support the NAND Flash Controller

Access to the NDFC memory mapped register is through the EBC. The base address of the NDFC registers is dependent on the memory map specified by the EBC Bank registers. For each NDFC chip enable ($\overline{\text{NFCE0:3}}$) attached to NAND Flash, the corresponding EBC Bank registers must be configured.

36.5.1 EBC0_BnAP

For the EBC bank or banks that are used to communicate with the NDFC, the EBC0_BnAP registers must be set to 0x018003C0.

When booting from NAND Flash, the NDFC uses EBC Bank 0 and therefore accesses the NDFC through the EBC using the default EBC0_B0AP configuration. After reset, EBC0_ B0AP is set to 0x7F8FFE80, which corresponds to maximum transfer timings for EBC Bank 0 (TWT = 255, CSN = OEN = WBN = WBF = 3, TH = 7). In addition to the wait states, ready enable is disabled, EBC0_B0AP[RE]=0. This EBC0_B0AP configuration does not provide adequate wait states or allow EBC_Ready to pace the transactions between the EBC and NDFC. To ensure the EBC does not prematurely access to the NDFC while reading the Initial Program Load, IPL, it is necessary to

enable AutoRead mode at reset time by setting SDR0_SDST2[NARE] = 1. AutoRead Mode when enabled with EBC Configuration Complete disabled, NDFC0_CR[EBCC] = 0, prevents EBC from counting wait states. In this mode the NDFC forces the EBC to supply enough wait states between accesses. (The reset default of EBC Configuration Complete is NDFC0_CR[EBCC] = 0).

The Secondary Program Load (SPL) must configure the EBC0_B0AP and NDCC0_CR (set EBC0_BOAP = 0x018003C0, set NDFC0_CR[EBCC] = 1 and set NDFC0_CR[ARE] = 0) before copying code from the NAND Flash to DDR memory or on-chip memory.

36.5.2 EBC0_CFG

Set EBC0_CFG[PTD] = 1. Disabling device paced time out forces the EBC to wait indefinitely for an internal ready signal from the NDFC.

Set EBC0_CFG[RTC] = 0b111. Ready Timeout Counter is set to the maximum of 2048 cycles. This prevents the EBC from ending normal NAND Flash cycles prematurely.

Settings of other EBC0_CFG fields do not affect how EBC communicates with NDFC.

36.5.3 EBC0_BxCR

The NDFC registers are accessed through the EBC in the EBC memory map. The base address in the EBC0_BxCR[BAS] is the base address for the NDFC registers.

The EBC bank size can be set to the smallest size since the NDFC only requires enough memory space for the NDFC registers and a 4KB linear access region. Set EBC0_BxCR[BS] = 0b000.

See *Section 36.6* for a description of the 4KB linear access region.

Set EBC0_BxCR[BU] = 0b11. The NDFC registers are read/write.

Set EBC0_BxCR[BW] = 0b10. The NDFC is a 32-bit EBC attached slave device.

36.5.4 Summary of EBC and NDFC Interface Configurations

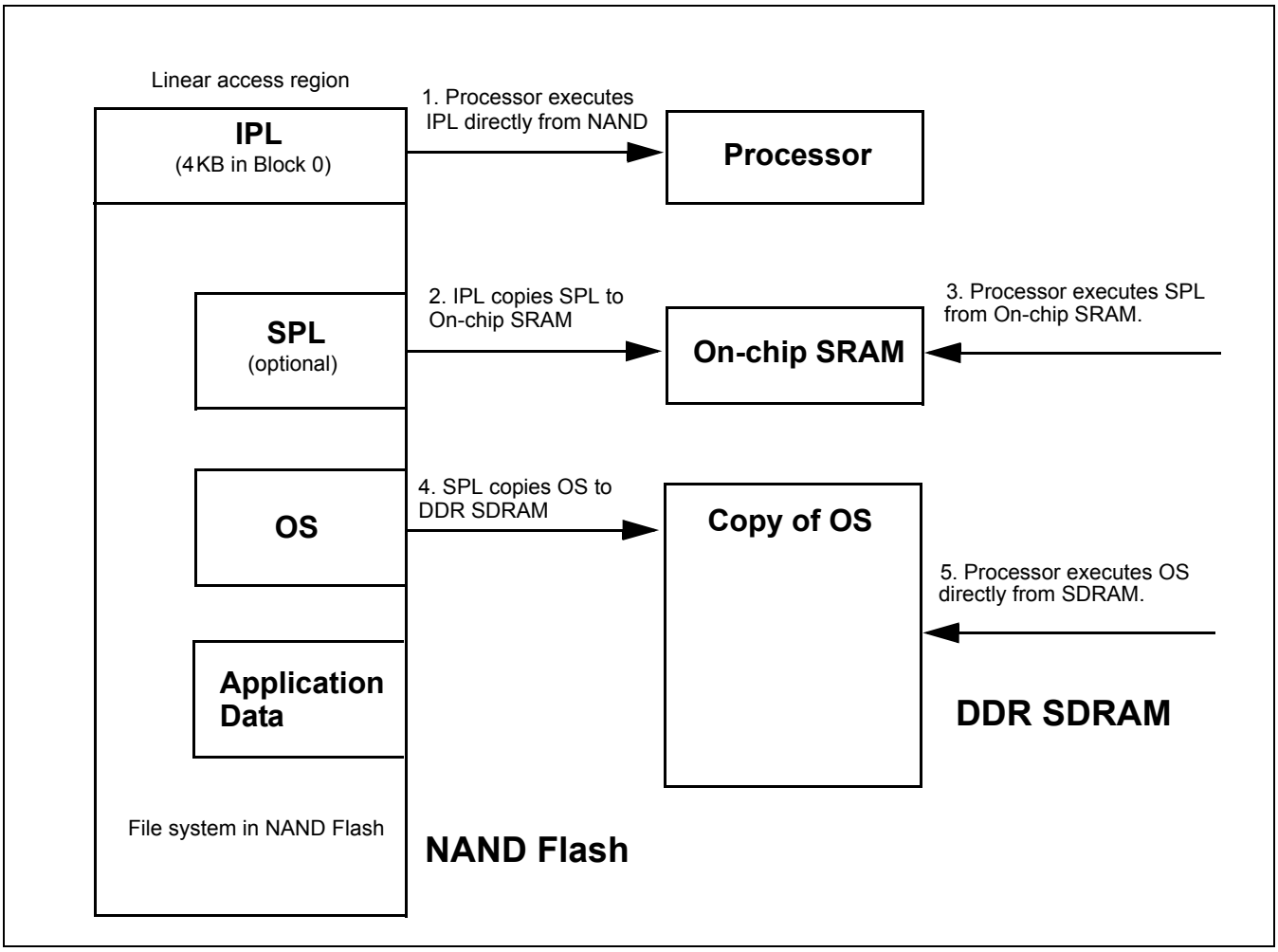
1. Configure EBC0_BxCR
   - set EBC0_BxCR[BAS] to the base address desired
   - set EBC0_BxCR[BS] = 0b000
   - set EBC0_BxCR[BU] = 0b11
   - set EBC0_BxCR[BW] = 0b10
2. Set EBC0_BnAP = 0x018003C0
3. Set EBC0_CFG = 0x78000000
4. Configure SDR0_CUST0
   - set SDR0_CUST0[MEN] = 0b10
   - set SDR0_CUST0[NE] = 1, to enable the NDFC controller
   - Enable the NDFC chip enables, SDR0_CUST0[NCG]
5. Configure the GPIO if using $\overline{\text{NFCE1:3}}$
6. Configure NDFC0_CR and NDFC0_BxCR

36.6 Booting from NDFC

When configured for booting, the NDFC's 4KB linear access region is the target of the PowerPC boot vector access at 0xFFFFFFFC. The 4KB linear access region is mapped to the last 4KB of memory, 0xFFFFF000–0xFFFFFFFF, and should contain the Initial Program Loader (IPL). The linear access region is located in Block 0 of the NAND Flash. Typically the linear region corresponds to pages 0, 1, 2, and 3 of memory with 512B pages or pages 0 and 1 of memory with 2KB pages depending on SDR0_SDSTP2[NBP]. The first page accessed in Block 0 depends on the page number pointed to by SDR0_SDSTP2[NBP].

The linear access region at boot time behaves from a software perspective as a 4KB ROM. When booting, auto-read mode is enabled, NDFC0_CR[ARE]. In this mode, the NAND controller generates the appropriate NAND Flash commands and address cycles. The format of the command and address cycles used in auto-read mode is selected by NDFC0_CR[ARAC].

Figure 36-2. IPL/SPL Boot Sequence Using Boot-from-NAND



Note: Block 0 must have a low error rate. Only use SLC NAND with Block 0 guaranteed when booting from NAND flash. The guarantee on Block 0 means Block 0 does not contain any stuck bits. Block 0 even with the guarantee is still affected by soft errors. Before selecting memory contact the memory vendor for the error rate.

Keep in mind that when booting directly from a NAND Flash device, there is no ECC correction or detection coverage on the data coming from NAND Flash while the IPL sequence is executing. This is because the instruction stream is directly fetched from the NAND Flash device and the ECC code that is stored in the extended/spare area is not read during IPL code fetch operations. The IPL code itself can use ECC to cover the rest of the SPL when loading the SPL and the operating system into on-chip SRAM or DDR memory. In order to use ECC to cover all portions of the code loaded from the NAND Flash, it is necessary to boot the IPL from a small NOR flash.

In order to boot from the NAND Flash, several parameters must be configured prior to booting by either Boot Option F in Table 9-3 on page 219 or Boot Option G or H. The Bootstrap Controller programs SDR0_SDSTP0:3 using the default settings of Option F or with settings read from an IIC serial ROM when using Option G or H.

Table 36-1 lists the boot configurations required for booting from NAND Flash. Column 1 contains the read only SDR0_SDSTP bit fields initialized by the Bootstrap Controller. Column 2 are the configuration straps for the NDFC and EBC. The values in the SDR0_SDSTP are used to initialize the registers in column 2. Column 3 are the EBC and NDFC that are initialized by the registers in column 2.

SDR0_CUST0[MEN, NCG, NE] must be set so that the NDFC signals and the NDFC are enabled.

SDR0_CUST0[NDRSC] is a 16-bit value that specifies the maximum number of clock cycles that the NDFC waits for NFRdyBusy (1 = ready) after a device reset command, before proceeding. The value used by Boot Option D is recommended.

SDR0_CUST1[NDRDC] is a 16-bit value that specifies the maximum number of clock cycles that the NDFC waits for NFRdyBusy (1 = ready) after a page read command, before proceeding. The worst case for the page read access time in many NAND Flash devices is 50 µs (specified in NAND Flash device data sheets as tR). The value used by Boot Option D is recommended.

Set SDR0_CP440[RL] = 0b10. RL = 0b10 configures the boot source as NDFC.

Set SDR0_EBC0[RW] = 0b10. The NDFC is internally a 32 bit EBC attached slave device; therefore, RW must be set to 0b10.

Some registers in the NDFC controller itself must be configured before booting. By virtue of having selected NDFC as a boot device, the chip automatically sets some bits in NDFC0_B0CR and NDFC0_CR. In particular, the timing values CRW, RWP, RWH, and RR are set to their maximum.

Set SDR0_SDSTP2[NBW] to the bus width of the NAND Flash.

Set SDR0_SDSTP2[NBAC] to the addressing mode and page size supported by the NAND Flash. This configuration is used by Auto-read mode to generate the correct address cycles when the IPL executes code from the 4KB linear region.

.

Table 36-1. Strap Register Configuration for Booting from NAND Flash

Read only Strap	Read/Write Strap	EBC and NDFC Bit Fields
SDR0_SDSTP1[RW] = 0b11	SDR0_EBC0[RW] = 0b11	EBC0_B0CR[BW] = 0b11
SDR0_SDSTP1[RL] = 0b10	SDR0_CP440[RL] = 0b10	N/A
SDR0_STP2[MEN] = 0b10	SDR0_CUST0[MEN] = 0b10	N/A
SDR0_SDSTP2[NE] = 1	SDR0_CUST0[NE] = 1	NDFC0_B0CR[EN] = 1
SDR0_SDSTP2[NBW]	SDR0_CUST0[NBW]	NDFC0_B0CR[SZ]
SDR0_SDSTP2[NBP] = 0b000	SDR0_CUST0[NBP] = 0b000	NDFC0_CR[RPG] = 0b000

Table 36-1. Strap Register Configuration for Booting from NAND Flash

SDR0_SDSTP2[NBAC]	SDR0_CUST0[NBAC]	NDFC0_CR[ARAC]
SDR0_SDSTP2[NARE] = 1	SDR0_CUST0[NARE] = 1	NDFC0_CR[ARE] = 1
SDR0_SDSTP2[NRB] = 0	SDR0_CUST0[NRB] = 0	NDFC0_CR[REN] = 0
SDR0_SDSTP2[NDRSC] = 0X8235	SDR0_CUST0[NDRSC] = 0X8235	N/A
SDR0_SDSTP2[NCG] = 0b1000	SDR0_CUST0[NCG] = 0b1000	N/A
SDR0_SDSTP2[NRDC] = 0X0D05	SDR0_CUST1[NRDC] = 0X0D05	N/A

36.6.1 AutoRead Mode

In the AutoRead mode (NDFC0_CCR[ARE] = 1), the NDFC automatically generates the page read command and page read address based on the read cycle address that is driven by the EBC. If the EBC read cycle's address is exactly +4 from the previous read cycle then the NDFC generally does not issue a new read command to the NAND Flash device, rather reads and return the next sequential piece of data. The exception to this +4 rule is when an access crosses the NAND Flash device's data page boundary. To prevent reading data in the device's spare area and returning this as valid data, the NDFC detects when an access crosses the device's data page size boundary and also treats this access as non-sequential.

Sequential read address detection allows the NDFC to avoid unnecessary read commands, and their corresponding page read access delay (tR), for code that is fetched sequentially from memory. EBC write cycles to the NDFC do not affect the sequential read address detection, so for example, the NDFC configuration registers can be written to update the configuration of NDFC0_CR to speed up external NAND Flash access timings. Also read cycles that are not inside the linear access region do not affect the sequential read address detection, so reads of the NDFC configuration or status registers are possible as well.

Note: AutoRead Mode is only used at boot time when executing the IPL directly from NAND Flash. It enables access to the 4KB linear access region.

The NDFC supports NAND Flash using AutoRead mode and Ready/Busy enabled when the device tWB is lower than

PerClk period x (NDFC0_B0CR[RWH] + 5).

In the case of boot from NAND Flash, the NDFC0_B0CR[RWH] default value is 8, which makes

tWB < (PerClk period x 13).

For PerClk = 100 MHz, the device tWB must be lower than 130 ns.

For PerClk = 83 MHz, the device tWB must be lower than 156 ns.

For PerClk = 66 MHz, the device tWB must be lower than 195 ns.

After boot, for better performance, the NDFC0_B0CR[RWH] value can be adjusted to be the first value immediately higher than tWB.

In auto-read mode, the number of command and address cycles is dependent on the NDFC0_CR[ARAC] configuration. *Table 36-2* contains the command and address sequence for each auto-read access cycle configuration. The commands and address driven on the NDFC data interface are described in Tables *36-3* and *36-4*. *Table 36-5* indicates how the row and column address cycles relate to the linear access region and the internal EBC address bus.

Table 36-2. Address/Commands Sequence Auto-Read Mode

SDR0_SDSTP[NBAC] SDR0_CUST0[NBAC], NDFC0_CR[ARAC]	Page Size	# CMD Cycles	# Address cycles	Order of Cycles
00	512B	1	3 cycles	<CMD0>,<column0>,<row0>,<row1>
01	512B	1	4 cycles	<CMD0>,<column0>,<row0>,<row1>,<row2>
10	2KB	2	4 cycles	<CMD0>,<column0>,<column1>,<row0>,<row1>,<CMD1>
11	2KB	2	5 cycles	<CMD0>,<column0>,<column1>,<row0>,<row1>,<row2>,<CMD1>

Table 36-3. Page Read Commands for an Auto-Read Cycle

SDR0_SDSTP[NBAC] SDR0_CUST0[NBAC] NDFC0_CR[ARAC]	Device Width	Page Size	<CMD0>	<CMD1>	Description
00, 01	8-bit	512B	0b0000000N	n/a	The N in 0b0000000N <CMD0> is driven by EBC_Addr23. To select either the 1st or 2nd 256-Bytes in a 512-B page size device, the distinction between a READ0 and a READ1 command is made with internal EBC_Addr23.
	16-bit	512B	0x00	n/a	A 16-bit wide device with a 512-B data page size corresponds to 256 halfword-sized columns, which can be all be reached with 1 column access cycle, and thus the READ1 command is unnecessary
10, 11	8/16-bit	2KB	0x00	0x30	2kB data page size devices are assumed to require a 2nd read command input after the address cycles to start the page read access in the device.

Table 36-4. Page Address for an Auto-Read Cycle

Cycle	Device Width	Page Size	<Address>	Description
<column0>	8-bit	512B/2KB	PerAddr24:31	LS byte address. Address driven internally to NAND Flash controller on EBC address bits EBC_Addr24:31.
	16-bit	512B/2KB	PerAddr23:30	LB halfword address. Address driven internally to NAND Flash controller on EBC address bits EBC_Addr23:31.
<column1>	8-bit	2KB	0b00000NNN	MS byte address for a 2KB page size device. NNN is driven internally to NAND Flash controller by EBC address bits EBC_Addr21:23.
	16-bit	2KB	0b000000NN	MS byte address for a 2KB (1K-halfwords) page size device. NN is driven internally to NAND Flash controller by EBC address bits EBC_Addr21:22.

Table 36-4. Page Address for an Auto-Read Cycle

Cycle	Device Width	Page Size	<Address>	Description
<row0>	8/16-bit	512B	0b0PPPPNNN	select one of eight 512B pages in Block m. PPPP driven with the page number specified by the bootstrap SDR0_SDSTP2[NBP]. NNN is driven internally to NAND Flash controller by EBC address bits EBC_Addr20:22.
		2KB	0b000PPPPN	select one of two 2KB pages in Block m. PPPP driven with the page number specified by the bootstrap SDR0_SDSTP2[NBP]. N is driven internally to NAND Flash controller by EBC address bits EBC_Addr20.
<row1>	8/16-bit	512B/2KB	0b00000000	
<row2>	8/16-bit	512B/2KB	0b00000000	

Table 36-5. Address Driven with Auto-Read Mode Enabled

Internal EBC address EBC_Addr0:31	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
NDFC Linear Access Region																				1	Linear Access Region											
NAND Device Address	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
512B page 8-bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PPPP				Row0			a*	Column0							
512B page 16-bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PPPP				Row0			Column0								-
2KB page 8-bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PPPP				R0	Col1			Column0							
2KB page 16-bit	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		PPPP				R0	Col1		Column0								-

PPPP driven with the page number specified by the bootstrap SDR0_SDSTP2[NBP].
a* is driven by EBC_Addr23. To select either the 1st or 2nd 256-Bytes in a 512- B page size device, the distinction between a READ0 and a READ1 command is made with internal EBC_Addr23.

36.6.2 Accessing NAND Flash

Access to NAND Flash is through the NDFC0_CMD, NDFC0_ADDR, and NDFC0_DATA registers. Writing NDFC0_CMD and NDFC0_ADDR registers generates command and address cycles and reading or writing the NDFC0_DATA register generates command and address cycles and reading or writing to the NDFC0_DATA register generates data access on the NDFC interface. The command written on the NDFC interface is the same as the value written to the NDFC0_CMD register. Refer to the NAND Flash data sheet for command and address cycles required for a particular NAND Flash.

36.6.2.1 Reading NAND Flash

The following example is a NAND Flash read and demonstrates four address cycles:

```
outbyte(NDFC0_CMD, command);
outbyte(NDFC0_CMD, command);    //read command
outbyte(NDFC0_ADDR, first_addr);
```

```
    outbyte(NDFC0_ADDR, second_addr);
    outbyte(NDFC0_ADDR, third_addr);
    outbyte(NDFC0_ADDR, fourth_addr);

//  Wait for address cycles to complete

    for (i=0; i < size; i++) // Size is the number of bytes in the page
    {
         destinationAddress[addr]=(inbyte(NDFC0_DATA));
         addr++;
     }
```

36.6.2.2 Writing NAND Flash

The following example is a NAND Flash write and demonstrates four address cycles:

```
    outbyte(NDFC0_CMD, command);
    outbyte(NDFC0_CMD, command);    //write command
    outbyte(NDFC0_ADDR, first_addr);
    outbyte(NDFC0_ADDR, second_addr);
    outbyte(NDFC0_ADDR, third_addr);
    outbyte(NDFC0_ADDR, fourth_addr);

//  Wait for address cycles to complete


    for (i=0; i < size; i++) // Size is the number of bytes in the page

    {

       outbyte(NDFC0_DATA , sourceAddress[i];

    }
```

36.7 ECC

The NDFC implements a Hamming Code Error Correction Code (ECC) that is compatible with SmartMedia format. This ECC is based on parity calculations of the read (incoming) or write (outgoing) data and consists of three bytes (22 bits) per 256B (2048 bits) data segment and provides single-bit error correction, double-bit error detection (SEC/DED) on the 256B block. The 22-bit ECC code is composed of 16-bit line parity and 6-bit column parity.

- Hamming ECC Algorithm is directly compatible with SmartMedia specification.
- ECC accumulated during page write and is available to software for programming after either 256B or 512B of page data write cycles are completed.
- ECC calculated during page read on incoming data and is available for comparison with ECC stored in device.
- Eight ECC registers accumulate calculated ECC data for each 256B block in an entire 2KB page-sized device.
- ECC hardware assist is mainly useful when reading entire pages.
- Software is responsible for making the stored versus calculated ECC comparison and performing any required bit error correction to the data block.

- The hardware ECC may be supplemented with stronger software ECC algorithm to improve coverage for Multi-Level-Cell (MLC) NAND Flash devices, where the failure of a single transistor can cause the loss of 2 bits of data.

For the SmartMedia format, the ECC code stored in the external NAND Flash device is always stored in the format shown in *Table 36-6*.

Table 36-6. ECC Code Assignment Table

I/O 7	I/O 6	I/O 5	I/O 4	I/O 3	I/O 2	I/O 1	I/O 0
P64	P64'	**P32**	P32'	**P16**	P16'	**P8**	P8'
P1024	P1024'	**P512**	P512'	**P256**	P256'	**P128**	P128'
P4	P4'	**P2**	P2'	**P1**	P1'	1	1
P8 to P1024 = Line/Row Parity P1 to P4 = Column Parity							

The NDFC algorithm for generating the 22-bit ECC is compatible with the requirements for SmartMedia devices. This algorithm handles the correction of single bit errors (correctable) and the detection of double bit errors (uncorrectable).

Note that if there are three or more bit errors in a page of read data, then the Hamming code may improperly determine that there is a single, correctable bit error and thus indicate that the software does a bit correction. This results in "mis-correction" of already bad data. If the application uses a NAND Flash device where more than two bit errors per 256B of page data is a possibility, then the software for that application should implement a stronger ECC algorithm, similar to Reed/Solomon code.

ECC is accumulated for every data byte that is either read or written by means of the NDFC0_DATA register or by means of the linear access region. After 256B of data have been read or written, the calculated ECC result is store in one of the NDFC0_ECCn registers (NDFC0_ECC0–NDFC0_ECC7) and the next NDFC0_ECCn register is selected. Then subsequent data reads/writes start accumulating a new ECC value and after 256B more data, that next result is stored and the next NDFC0_ECCn register is again selected. This process repeats until a calculated ECC result is stored into NDFC0_ECC7, which indicates that 2KB of data has been written (without any ECC reset occurring). After NDFC0_ECC7 is update, the ECC calculation is halted for subsequent reads/writes, and NDFC0_ECC0:7 registers are not updated again until a 16-bit wide NAND Flash device (NDFC0_CR[SZ] = 1) causes two bytes of "accumulation" for the ECC calculations.

To check the ECC on a block of read data is a simple matter of performing a bit wise XOR function between the calculate ECC (read from NDFC0_ECC0-NDFC0_ECC7) and the store ECC (read from spare/extended data area of external NAND Flash device). The results of this XOR gives the ECC syndrome.

Table 36-7. Hamming ECC Syndromes

XOR Result	Meaning
22-bits all 0	No Error
Exactly 11 bits are 1	Correctable Error
1 bit is 1	ECC Code Error
Anything Else	Uncorrectable Error

No Error

When the XOR result value has all 22-bits with the value 0, this indicates that the stored ECC which was read from the external NAND Flash device and the calculated ECC read from the NFECC0–NFECC7 register are identical and there is not error in the page data and no error in the stored ECC data.

Correctable Errors

For correctable errors, the XOR result value has the property where all parity bit pairs (e.g. P8 and P8) have opposite values (1 and 0 or 0 and 1), with exactly 11 bits of the 22-bit code set to 1. By extracting the XOR result's parity bits (P1024, P512, P256, P128, P64, P32, P16, P8, P4, P2, P1), the exact bit position of the single bit error can be determined.

(P1024, P512, P256, P128, P64, P32, P16, P8) = Offset of byte containing single bit error (P4, P2, P1)=Bit number of single bit error.

> Example: If XOR Results is (**P1024**, P1024, **P512**, P512, ..., **P2**, P2, **P1**, P1) = 0b**0**1**1**0**0**1**0**1**1**0**1**0**1**0**0**1**0**1**1**0**1**0**0**1, this result has the correct properties to qualify as a single, correctable bit error. Extracting the parity bits (**P1024**, **P512**, ...**P16, P8**) (**P4, P2, P1**) = (01001100) (110). This result means that at byte offset 76 (4Ch), bit 6 should be flipped to correct the single bit error.

ECC Code Errors

A single bit set to 1 in the XOR result value indicates that the single bit error occurred in the stored ECC code and not in the page of read data.

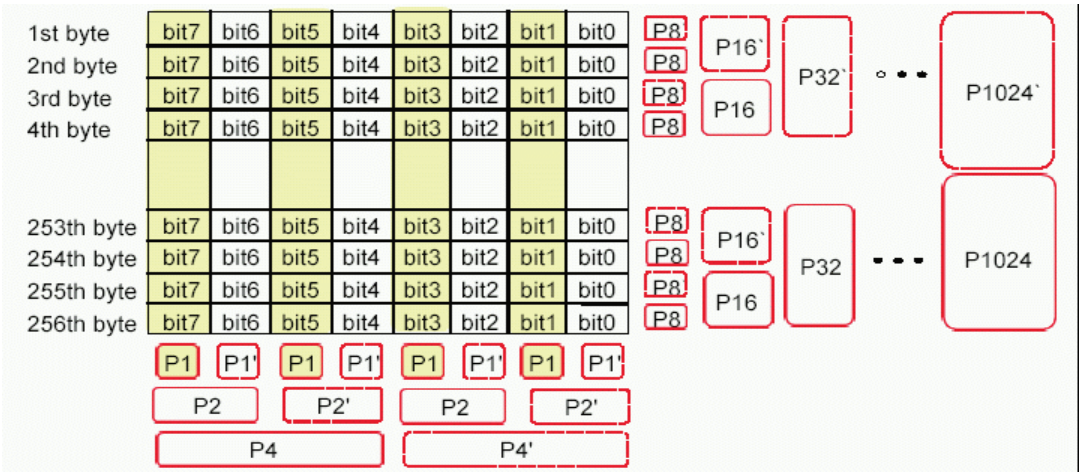
Uncorrectable Errors

If the XOR result value is non-zero and does not meet the criteria for a correctable error or ECC code error, then the error should be considered uncorrectable as two or more bits of the page data were corrupted and the Hamming ECC algorithm cannot determine which bytes/bits are faulty. In this case, the entire page of read data should be discarded.

36.7.1 Column Parity

The NDFC ECC generator calculates 6-bits of Column parity. These parity bits are called CP00–CP05 and correspond to (P1, P1', P2, P2', P4, P4').

Figure 36-3. Column Parity Generation



P1 = bit7 ⊕ bit5 ⊕ bit3 ⊕ bit1 ⊕ P1
P2 = bit7 ⊕ bit6 ⊕ bit3 ⊕ bit2 ⊕ P2
P4 = bit7 ⊕ bit6 ⊕ bit5 ⊕ bit4 ⊕ P4

P1' = bit6 ⊕ bit4 ⊕ bit2 ⊕ bit0 ⊕ P1'
P2' = bit5 ⊕ bit4 ⊕ bit1 ⊕ bit0 ⊕ P2'
P4' = bit3 ⊕ bit2 ⊕ bit1 ⊕ bit0 ⊕ P4'

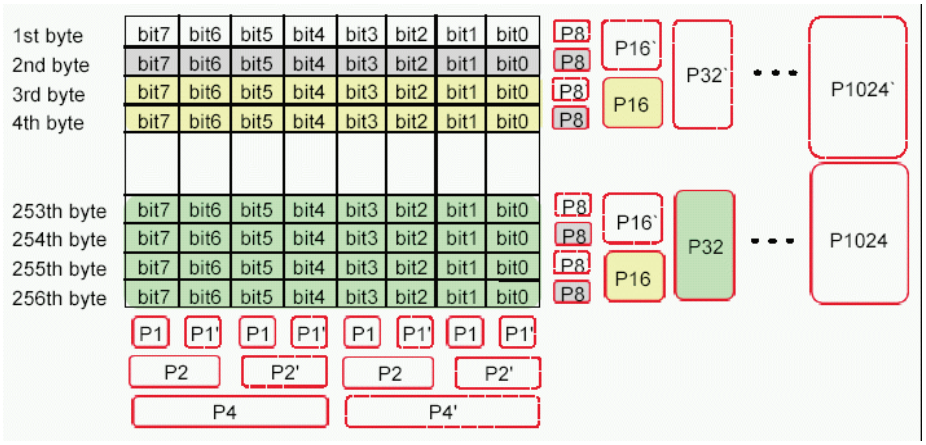
⊕ = XOR

Column parity is accumulated for each byte of data that is read or written to the external NAND Flash device when ECC is enabled.

36.7.2 Line/Row Parity

The NDFC ECC generator calculates 16-bits of Line/Row parity. These parity bits are called LP00 - LP15 and correspond to (P8, P8', P16, P16', P32, P32', ... P1024, P1024').

Figure 36-4. Line/Row Parity Generation



P8 = (even byte) = bit7 ⊕ bit6 ⊕ bit5 ⊕ bit4 ⊕ bit3 ⊕ bit2 ⊕ bit1 ⊕ bit0 ⊕ P8
P8' = (odd byte) = bit7 ⊕ bit6 ⊕ bit5 ⊕ bit4 ⊕ bit3 ⊕ bit2 ⊕ bit1 ⊕ bit0 ⊕ P8'
etc...

⊕ = XOR

36.7.3 How to generate line parity (LP00-LP15)

The line parity, which is recalculated for each data byte, depends on the data bytes which have already been input. If the total sum of the bits of 1 byte of data is 0, the line parity does not change when it is recalculated.

The sum of the bits in 1 byte of data is:

D_{all} = D0 XOR D1 XOR D2 XOR D3 XOR D4 XOR D5 XOR D6 XOR D7

Sixteen line parity bits need to be computed (LP0–LP15) from 256B of data. This is done by initializing all line parity bits to 0, reading in each byte, computing the byte sum bit (D_{all}), and adding D_{all} to the line parity bits when they apply. This is done by using an 8-bit counter and using the appropriate bit as a type of mask. The counter is incremented by 1 for each new byte of data.

LP00 = LP00 XOR $\overline{\text{Counter\_bit0}}$ AND Dall, LP01 = LP01 XOR $\overline{\text{Counter\_bit0}}$ AND Dall
LP02 = LP02 XOR $\overline{\text{Counter\_bit1}}$ AND Dall, LP03 = LP03 XOR $\overline{\text{Counter\_bit1}}$ AND Dall
LP04 = LP04 XOR $\overline{\text{Counter\_bit2}}$ AND Dall, LP05 = LP05 XOR $\overline{\text{Counter\_bit2}}$ AND Dall
LP06 = LP06 XOR $\overline{\text{Counter\_bit3}}$ AND Dall, LP07 = LP07 XOR $\overline{\text{Counter\_bit3}}$ AND Dall
LP08 = LP08 XOR $\overline{\text{Counter\_bit4}}$ AND Dall, LP09 = LP09 XOR $\overline{\text{Counter\_bit4}}$ AND Dall
LP10 = LP10 XOR $\overline{\text{Counter\_bit5}}$ AND Dall, LP11 = LP11 XOR $\overline{\text{Counter\_bit5}}$ AND Dall
LP12 = LP12 XOR $\overline{\text{Counter\_bit6}}$ AND Dall, LP13 = LP13 XOR $\overline{\text{Counter\_bit6}}$ AND Dall
LP14 = LP14 XOR $\overline{\text{Counter\_bit7}}$ AND Dall, LP15 = LP15 XOR $\overline{\text{Counter\_bit7}}$ AND Dall

For example, LP00 should be calculated from all even data bytes and LP01 should be calculated from all odd data bytes. Bit 0 of the counter is ANDed with D_{all} to enable or disable whether D_{all} contributes to the line parity summation.

Thus, according to the line parity generation formulas as:

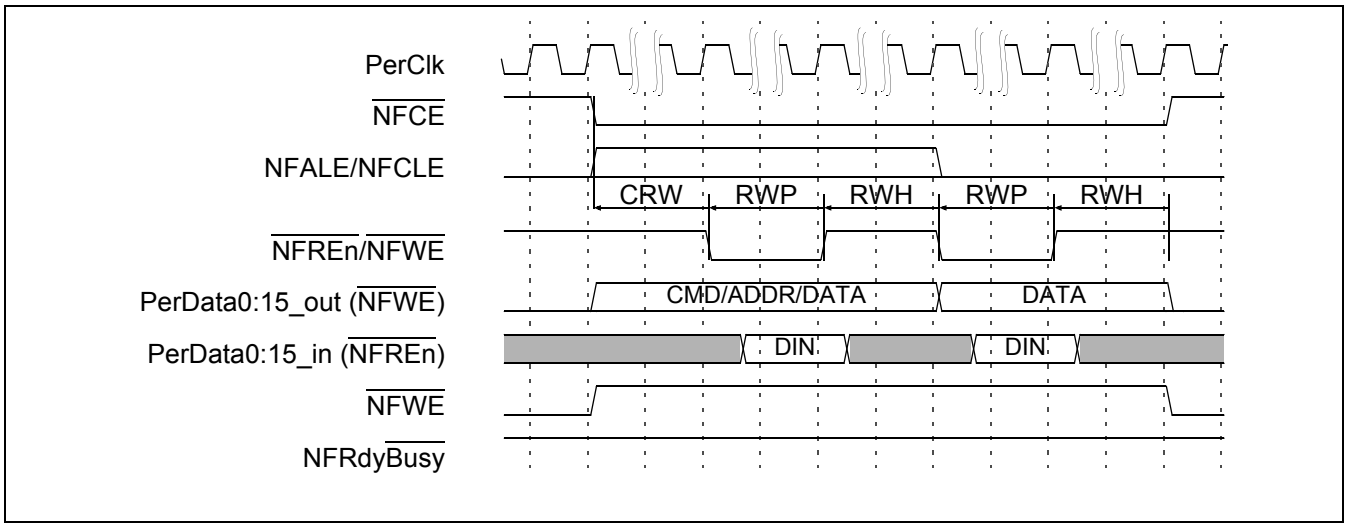
for LP00: 1st,3rd,5th,7th ... data bytes used (counter = 0, 2, 4, 6...)
for LP01: 2nd,4th,6th,8th ... data bytes used (counter = 1, 3, 5, 7...)
for LP02: 1st,2nd,5th,6th ... data bytes used (counter = 0, 1, 4, 5 ...)
for LP03: 3rd,4th,7th,8th ... data bytes used (counter = 2, 3, 6, 7...)
and so on.

36.8 External Device Cycles

The bus timing on the NAND Flash interface is programmable on a per chip enable basis using the timing parameters in the NFxCR registers. The following figures show how the timing parameters CRW, RWP, and RWH affect timing of various NAND Flash accesses.

- The NFALE and NFCLE outputs are active for CRW + RWP + RWH cycles as shown in *Figure 36-5*.
- When both NFALE and NFCLE are low, the DATA cycle can continue for additional RWP + RWH cycles until the transfer is satisfied. For example, a 32-bit word transfer from the EBC results in CRW + 4(RWP + RWH) cycles on the external bus. $\overline{\text{NFREn}}$ or $\overline{\text{NFWE}}$ are active only during each of the RWP cycle times.

Figure 36-5. Basic External Device Timing Controls



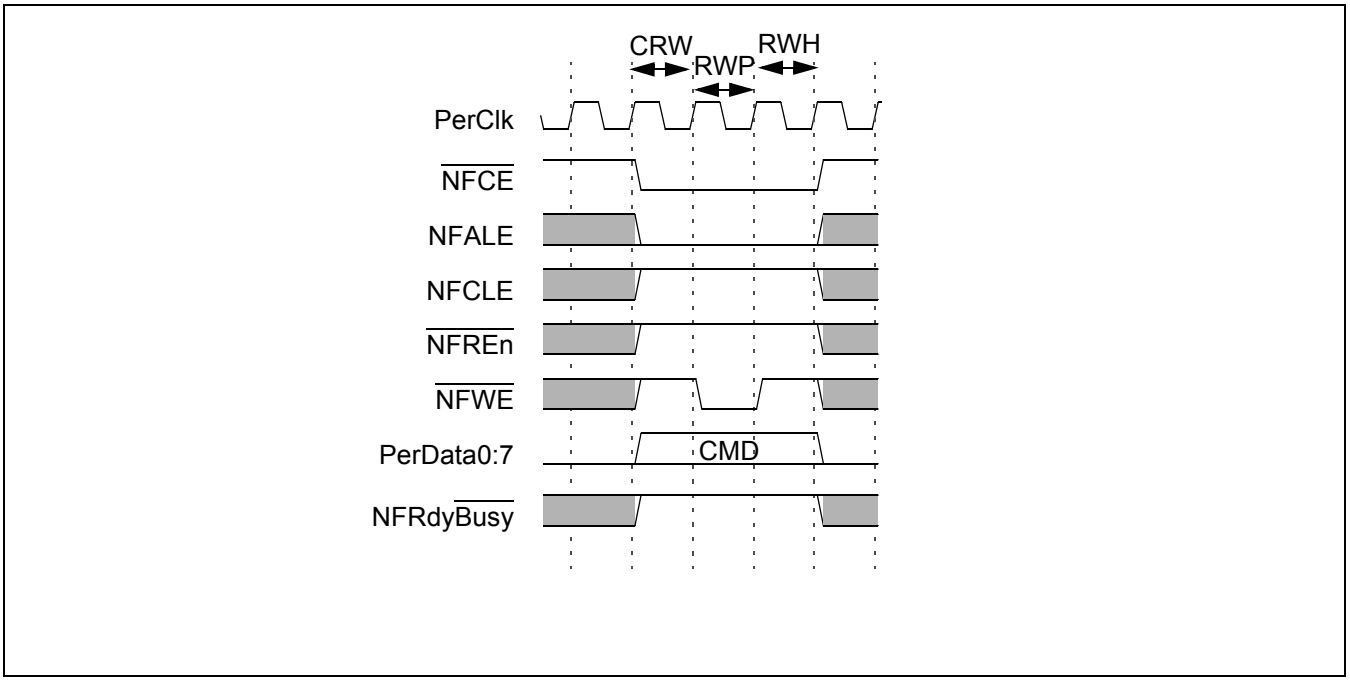
NFxCR[CRW] 0–7 Cycles clocks between NFCE/NFALE/NFCLE asserted and NFREn/NFWE asserted. When CRW = 0, NFREn/NFWE are asserted at the same time as NFCE/NFALE/NFCLE.

NFxCR[RWP] 1–8 Cycles Clocks that NFREn/NFWE stay active (Active Pulse width).

NFxCR[RWH] 1–8 Cycles Clocks that NFREn/NFWE stay inactive (Data Hold time).

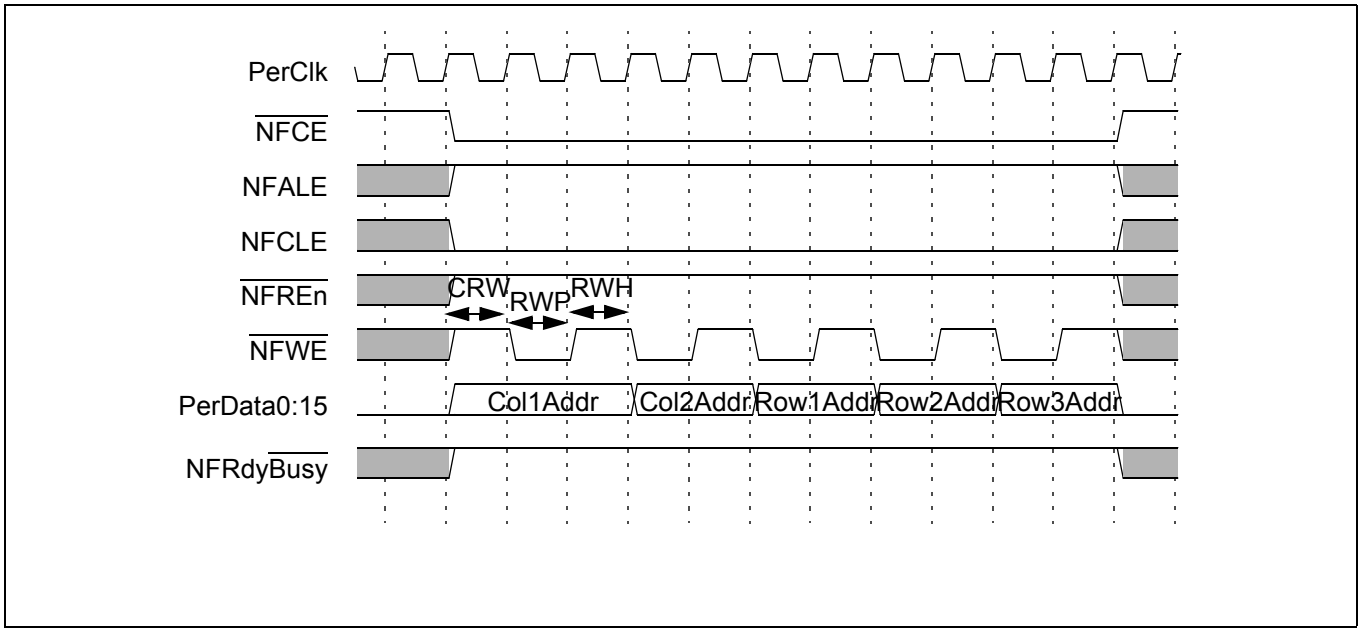
36.8.1 Command Cycle

Figure 36-6. Command write cycle



36.8.2 Address Write Cycle

Figure 36-7. Address Cycle



36.8.3 Basic Nand Flash Device Read Cycle

Figure 36-8. Single (raw) Read Cycle

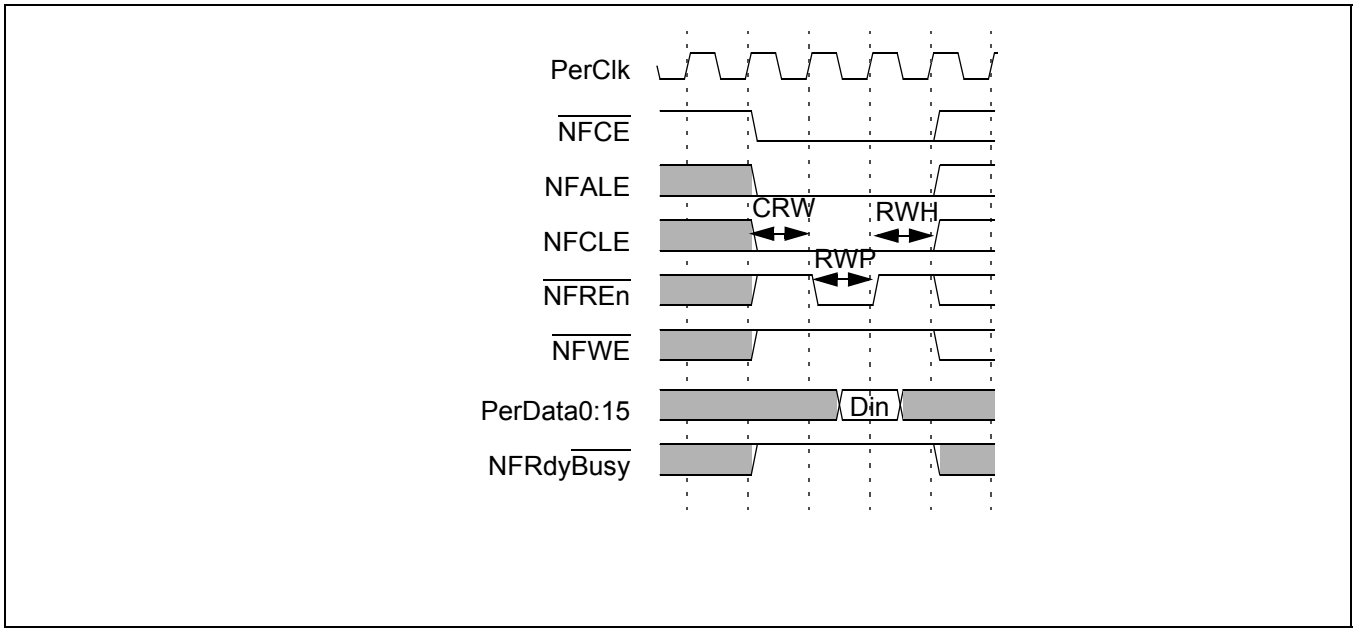
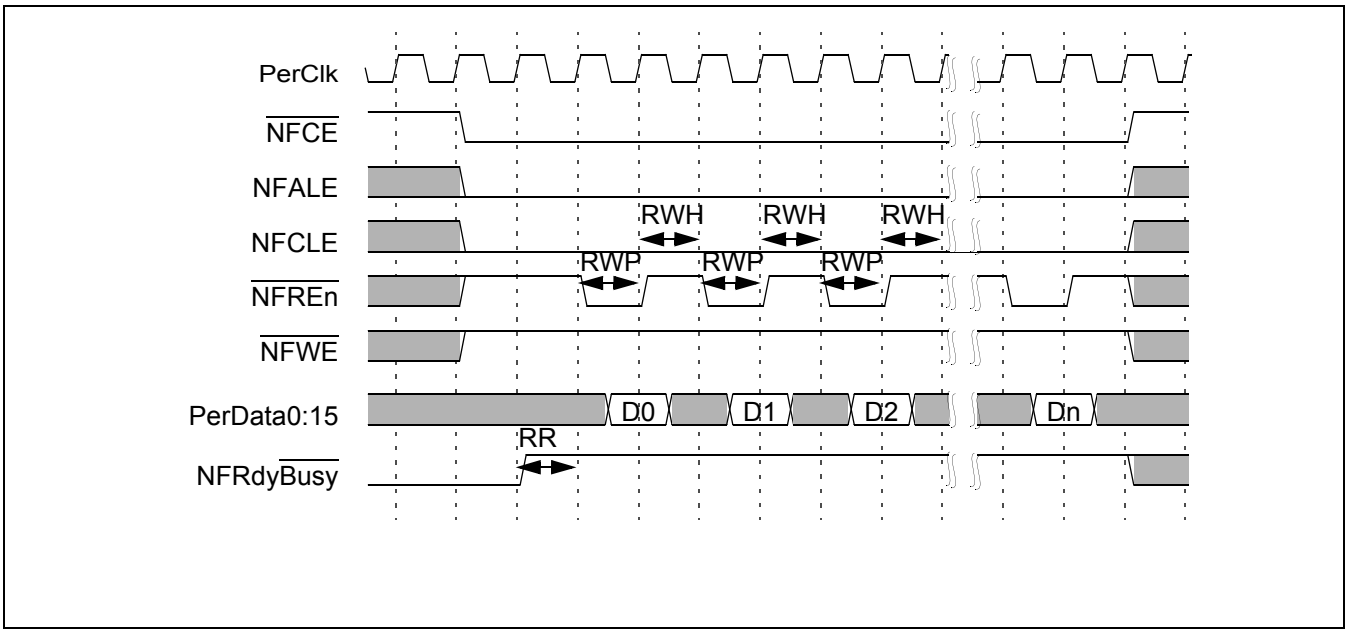


Figure 36-9. Burst (Page) Read Cycle



36.8.4 *B*asic Nand Flash Device Write Cycle

Figure 36-10. Single (raw) Write Cycle

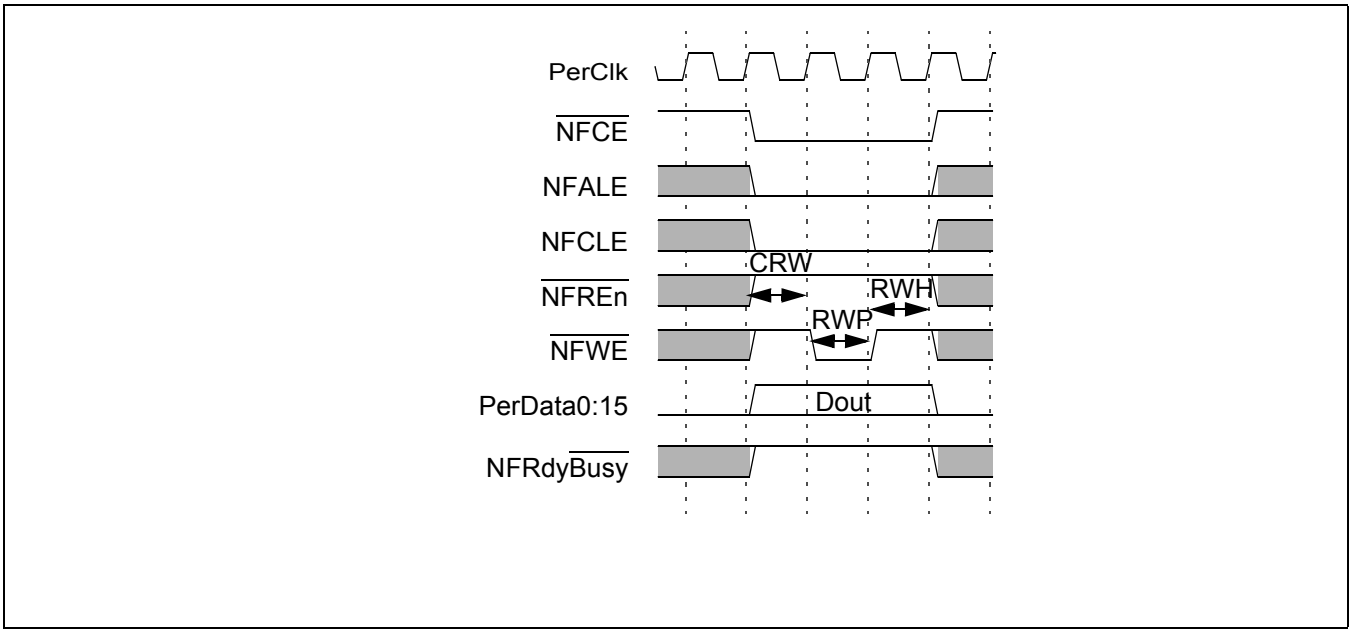
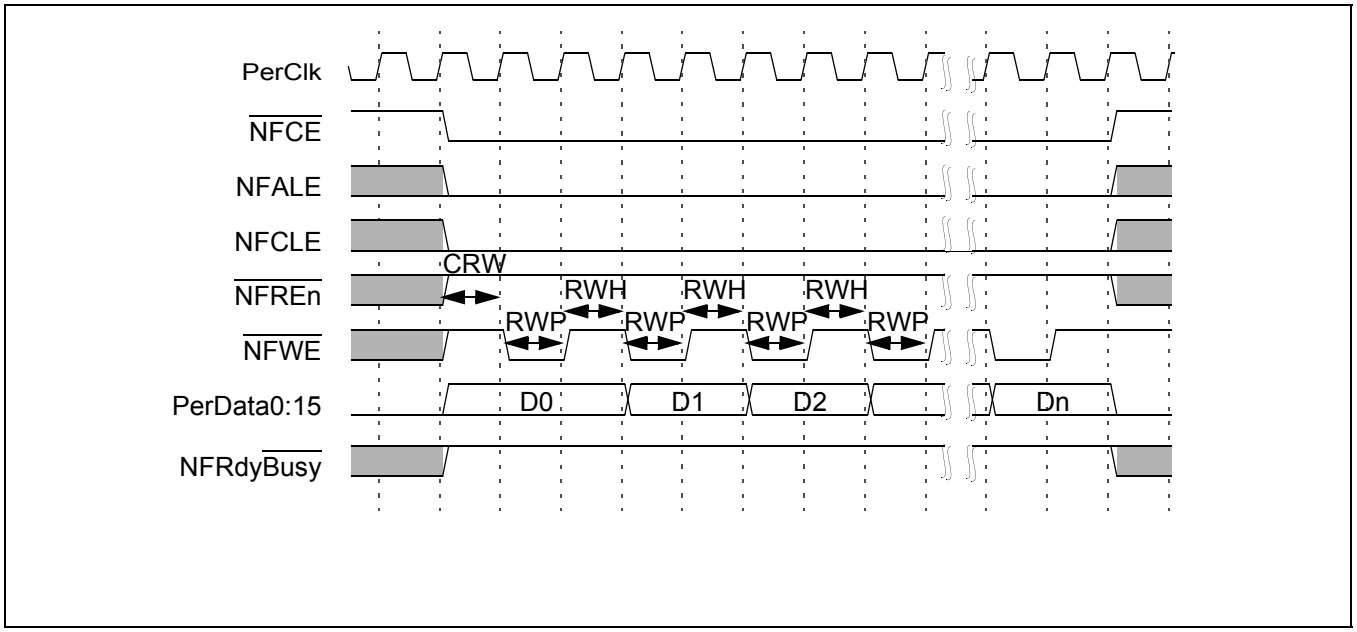


Figure 36-11. Burst (Page) Write Cycle



36.9 NDFC Registers

The NAND Flash controller provides one dedicated configuration register for each NAND Flash device chip selected (NFCE0:3). There are also 15 other common registers that are shared across all devices and are used to configure or monitor status of the common aspects of the NDFC.

36.9.1 Memory Map

This section lists all of the NDFC registers and describes how they are accessed.

The NDFC is relocatable within the EBC address space. See *External Bus Controller* on page 847 for details on how to program the base address and region size and assign the NDFC to one of the EBC banks. Addresses in *Table 36-8* are offsets to the EBC base address, EBC0_BnCR[BAS]. For example EBC0_B1CR[BAS] = 0xC00, software accesses the NDFC0_ADDR register at OPB address 0xC0000004. On PLB4, the 36 bit address for NDFC0_ADDR is 0x4C0000004.

Table 36-8. NAND Flash Controller Memory Map

Mnemonic	Register	Address Offset	Access	Page
NDFC0_CMD	NDFC Command Register	0x0000	R/W	1281
NDFC0_ADDR	NDFC Address Register	0x0004	R/W	1280
NDFC0_DATA	NDFC Data Register	0x0008	R/W	1281
NDFC0_ECC0	NDFC ECC Register 0	0x0010	R	1282
NDFC0_ECC1	NDFC ECC Register 1	0x0014	R	1282
NDFC0_ECC2	NDFC ECC Register 2	0x0018	R	1282
NDFC0_ECC3	NDFC ECC Register 3	0x001C	R	1282
NDFC0_ECC4	NDFC ECC Register 4	0x0020	R	1282
NDFC0_ECC5	NDFC ECC Register 5	0x0024	R	1282
NDFC0_ECC6	NDFC ECC Register 6	0x0028	R	1282
NDFC0_ECC7	NDFC ECC Register 7	0x002C	R	1282
NDFC0_B0CR	NDFC Bank Configuration Register 0	0x0030	R/W	1283
NDFC0_B1CR	NDFC Bank Configuration Register 1	0x0034	R/W	1283
NDFC0_B2CR	NDFC Bank Configuration Register 2	0x0038	R/W	1283
NDFC0_B3CR	NDFC Bank Configuration Register 3	0x003C	R/W	1283
NDFC0_CR	NDFC Configuration Register	0x0040	R/W	1284
NDFC0_SR	NDFC Status Register	0x0044	R	1286
NDFC0_HWCTL	NDFC Direct Hardware Control Register	0x0048	R/W	1286
NDFC0_REVID	NDFC Revision ID Register	0x0050	R	1288
na	Linear Data Access Region	0x1000 - 0x1FFF	R/W	na

36.9.2 NDFC Address Register (NDFC0_ADDR)

The NAND Flash Address register (NDFC0_ADDR) is an external hardware access register. When a write is made to this register, the NFALE signal is asserted and a write cycle is performed to the external device.

External device cycles are only performed when the currently selected bank, as programmed in the NDFC0_CR[BS] field, is enabled in its bank configuration register, NDFC0_CR[EN]. When the currently selected bank is not enabled, writes and reads to this register simply store and return stored data, respectively.

This register can be written with a transfer size of byte, half word or word.

Reads of this register only return the data stored in the NDFC0_ADDR register. No external device cycle is performed for a read of this register. Unused bits return a value of 0.

Figure 36-12. NAND Flash Address Register (NDFC0_ADDR)			
0:7	ADDR	Nand Flash Command	
8:31		Reserved	

36.9.3 NDFC Command Register (NDFC0_CMD)

The NAND Flash Command register (NDFC0_CMD) is an external hardware access register. When a write is made to this register, the NFCLE signal is asserted and a write cycle is performed to the external device.

External device cycles are only performed when the currently selected bank (as programmed in the NDFC0_CR[BS] field) is enabled in its bank configuration register, NDFC0_BnCR[EN]. When the currently selected bank is not enabled, writes and reads to this register simply store and return stored data, respectively.

This register can be written with a transfer size of byte, half word or word.

Reads of this register only return the data stored in the NDFC0_CMD register. No external device cycle is performed for a read of this register. Unused bits return a value of 0.

Figure 36-13. NAND Flash Command Register (NDFC0_CMD)			
0:7	CMD	Nand Flash Command	
8:31		Reserved	

36.9.4 NDFC Data Register (NDFC0_DATA)

The NAND Flash Data register (NDFC0_DATA) is an external hardware access register. When a write is made to this register, one or more write cycles are performed to the external device. When a read is made to this register, one or more read cycles are performed to the external device.

External device cycles are only performed when the currently selected bank, as programmed in the NDFC0_CR[BS] field, is enabled in its bank configuration register, NDFC0_CR[EN]. When the currently selected bank is not enabled, writes and reads to this register simply store and return stored data, respectively.

This register can be written with a transfer size of byte, half word or word.

Note that a byte size read transfer to a 16-bit NAND Flash device (NDFC0_CR[SZ]=1) results in the loss of the upper half of the data returned from the device. This condition is detected and causes the NDFC0_SR[RDL] bit to be set.

The total number of external accesses performed for each transfer size is dependent on the selected bank's device width.

Figure 36-14. NAND Flash Data Register (NDFC0_DATA)			
0:31	DATA	Nand Flash Data	

Table 36-9 and *Table 36-10* describe the data input and out ordering, respectively.

Table 36-9. Data Input Ordering to Read Data on EBC

Device Size	Read size	External Cycle #	PerData0:7	PerData08:15	Internal EBC Data bus (0:31)[1]
16	Word (4 bytes)	1 2	<aa> <cc>	<bb> <dd>	<**aabbccdd**>
	Half Word (2 bytes)	1	<aa>	<bb>	<**aabb**aabb>
	Bytes (1)[2]	1	<aa>	<bb>	<**aa**aaaaaa>
8	Word (4 bytes)	1 2 3 4	<aa> <bb> <cc> <dd>	<XX> <XX> <XX> <XX>	<**aabbccdd**>
	Half Word (2 bytes)	1 2	<aa> <bb>	<XX> <XX>	<**aabb**aabb>
	Bytes (1)	1	<aa>	<XX>	<**aa**aaaaaa>

Notes:
1. Reads of less than a word returns data mirrored onto all possible byte lanes.
2. A byte size read from a 16-bit device results in loss of data returned from the device. This sets the NDFC0_STAT[RDL] flag.

Table 36-10. Write Data to Output Ordering

Device Size	Internal EBCdata bus(0:31)	Write size	External Cycle #	PerData0:7	PerData08:15
16	<aabbccdd>	Word (4 bytes)	1 2	<aa> <cc>	<bb> <dd>
	<aabb>	Half Word (2 bytes)	1	<aa>	<bb>
	<aa>	Bytes (1)	1	<aa>	<XX>
8	<aabbccdd>	Word (4 bytes)	1 2 3 4	<aa> <bb> <cc> <dd>	<ZZ> <ZZ> <ZZ> <ZZ>
	<aabb>	Half Word (2 bytes)	1 2	<aa> <bb>	<ZZ> <ZZ>
	<aa>	Bytes (1)	1	<aa>	<ZZ>

36.9.5 NAND Flash ECC Calculation Registers (NDFC0_ECC0–NDFC0_ECC7)

The NAND Flash ECC Calculation Registers (NDFC0_ECC0–NDFC0_ECC7) are for reading the hardware generated ECC values for each 256B of page read or page write data.

These registers cannot be written. They can be cleared using the NDFC0_CR[ECR] (ECC Reset) bit. Reads of the NDFC0_ECC0:7 registers only return the ECC data stored in the register and do not affect the calculation or delete the ECC data.

Figure 36-15. NAND Flash ECC Calculation Register (NDFC0_ECC0–NDFC0_ECC7)			
0:7	BC	Byte Counter	
8:15	LP0	Line/Row Byte 0 ECC Calculated Result 8 - P64 9 - P64' 10 - P32 11 - P32' 12 - P16 13 - P16' 14 - P8 15 - P8 '	
16:23	LP1	Line/Row Byte 1 ECC Calculated Result 16 - P1024 17 - P1024' 18 - P512 19 - P512' 20 - P256 21 - P256' 22 - P128 23 - P128'	
24;31	CP	Column (bit offset) Calculated Result 24 - P4 25 - P4' 26 - P2 27 - P2' 28 - P1 29 - P1' 30 - 1 31 - 1	

36.9.6 NDFC Bank Configuration Registers (NDFC0_B0CR–NDFC0_B3CR)

The NAND Flash Bank Configuration Registers (NDFC0_B0CR–NDFC0_B3CR) are for enabling and setting access parameters for each of the external banks of NAND Flash.

This register can be written with a transfer size of byte, half word or word.

Figure 36-16. NAND Flash Bank Configuration Registers (NDFC0_B0CR–NDFC0_B3CR)			
0	EN	Bank Enable 0 Bank access disabled 1 Bank access enabled	When the NDFC0_ADDR, NDFC0_CMD, NDFC0_DATA, or Linear Data Access Region are accessed and the currently selected bank is disabled, the NDFC will not perform an external bus cycle, but rather just store the data into the appropriate register. NDFC0_CR reset default comes from SDR0_SDST2[NE].
1	CED	CE# style 0 CE# will remain asserted/active until ECC Reset or switching bank selects via NDCF0_CR 1 CE# is Don't Care style and will deassert when external access cycle is completed	When cleared, the selected bank's CE# signal is asserted when the first access is made to NDFC0_CMD, NDFC0_ADDR, or NDFC0_DATA registers or the Linear Data regions and will remain asserted until a CE# reset is performed (NDFC0_CR[CER]) or the active bank is switched (NDFC0_CR[BS]). NDFC0_CR reset default is always 0.
2:3		Reserved	

4	SZ	Bank Size 0 External Device is 8-bit, attached to PerData00:07 1 External Device is 16-bit, attached to PerData00:15	If SDR0_SDST2[NE] = 1, NDFC0_CR reset default comes from SDR0_SDSTP2[NBW].
5:16		Reserved	
17:19	CRW	CE# low to RE#/WE# starting (0-7)	Setting is based on PerClk cycles If SDR0_SDST2[NE] = 1, NDFC0_CR reset default = 0b111.
20		Reserved	
21:23	RWP	RE#/WE# low pulse time (1-8)	Setting is based on PerClk cycles If SDR0_SDST2[NE] = 1, NDFC0_CR reset default = 0b111.
24		Reserved	
25:27	RWH	RE#/WE# high pulse time (1-8)	Setting is based on PerClk cycles If SDR0_SDST2[NE] = 1, NDFC0_CR reset default = 0b111.
28		Reserved	
29:31	RR	RDY_BSY# high to RE# starting (1-8)	Setting is based on PerClk cycles If SDR0_SDST2[NE] = 1, NDFC0_CR reset default = 0b111. **Note:** This value only has an effect for read accesses in the linear region and only when NDFC0_CR[REN] is set.

36.9.7 NDFC Configuration Register (NDFC0_CR)

The NAND Flash Configuration Register (NDFC0_CR) is for enabling and setting access parameters for the entire NDFC.

This register can be written with a transfer size of byte, half word or word.

Figure 36-17. NAND Flash Configuration Register (NDFC0_CR)

0	CER	CE# Reset 0 no action 1 Reset any active Chip Enable output	Setting this bit to 1 deasserts an active CE# output. This is only necessary when a bank is configured with NDFC0_CR[CER] = 0 and the CE# signal would otherwise stay asserted.
1	ERC	ECC Reset 0 no action 1 Reset ECC	Setting this bit to 1 resets the ECC counter(s)/ calculation(s). **Note:** Writes to this bit are not actually stored in the register, but rather only serve to perform the reset function.
2	RIE	Interrupt Enable on Device Ready 0 No Interrupt output 1 Interrupt pulse generated when RDY/BSY# goes from 0 -> 1, indicating device has just completed some internal operation.	The Interrupt pulse is generated on the NDFC_IRQ function output and is four PerClk wide. Typically this means that the interrupt controller must detect this signal with an edge-sensitive input.

3	REN	Enable wait for Ready in reads from Linear Region 0 NDFC ignores value of NFRdyBusy when accessing the linear data region 1 NDFC waits for NFRdyBusy == 1 (ready) before allowing read cycles to occur in the linear data region.	Setting this bit also allows the NDFC0_BxCR[RR] timer to be activated when a read is performed in the linear region and the current, latched state of NFRdyBusy = 0 (busy). The NDFC0_BxCR[RR] counter counts when NFRdyBusy goes from 0 (busy) to 1 (ready) before allowing a read cycle to the external device. Clearing this bit means that software must have insured that the device is ready prior to accessing the external device. Clearing this bit is required since the NFRdyBusy signal is multiplexed with other signals. This bit can be strapped at reset time via SDR0_SDSTP2[NRB]. **Note:** This setting also affects AutoRead mode because accesses at reset time for AutoRead mode are sent to the linear access region.
4		Reserved	
5	ARE	Auto-Read Enable 0 Reads to Linear Region only generate read cycle to the external NAND Flash. 1 Reads to Linear Region can generate automatic page read commands with the address determined from the EBC read address and then perform a read cycle to the external NAND Flash.	When set, the address of reads to the Linear region is compared to the previous read address and if the new address is "non-sequential" (that is, it is not <prev_addr>+4), then an automatic "page read" command is performed. The address of the page read command is determined from the new address following the rules outlined in *AutoRead Mode* on page 1268.
6:7	BS	Bank Select 00 Select Bank on CE[0] 01 Select Bank on CE[1] 10 Select Bank on CE[2] 11 Select Bank on CE[3]	Changing the Setting from 0 to 3; setting determines which bank is "selected" for subsequent hardware accesses. Changing the Bank Select setting will also deassert any active chip enable (the same as writing NDFC0_CR[CER] = 1).
8:17		Reserved	
18:19	ARAC	Auto-Read mode Address Cycles 00 3 Addr. Cycles, 1 Col. + 2 Row (512 page size) 01 4 Addr. Cycles, 1 Col. + 3 Row (512 page size) 10 4 Addr. Cycles, 2 Col. + 2 Row (2k page size) 11 5 Addr. Cycles, 2 Col. + 3 Row (2k page size)	Used when AutoRead mode is enabled. This value determines how many address cycles are performed during the setup to configure the external device for reading the IPL from Bank 0. This value is preloaded from the SDR0_SDSTP2[NBAC] strapping input during reset. See *AutoRead Mode* on page 1268.
20:23	RPG	Auto-Read mode base Page value	Used when AutoRead mode is enabled. This value determines the base page address that an automatically generated page read command will output to the NAND Flash during AutoRead. This value is preloaded from the SDR0_SDSTP2[NBP] strapping input during reset. If this value is changed from it's strapped, default value, the software must take care to insure that the next AutoRead mode access made to the NDFC linear region is non-sequential so that the new page address value is sent to the external NAND Flash device. This is because writing to the NDFC0_CR register by itself does not cause the NDFC to assume that the ARPG value has changed. See *AutoRead Mode* on page 1268.
24:28		Reserved	
29	EBCC	EBC Configuration Completed 0 Gate clocks to the EBC to pace transactions when in AutoRead mode. Only used when booting from NAND Flash. 1 Never gate clocks to the EBC. Required setting for normal operation.	Immediately after reset, the EBC PerReady signal is not enable. To pace transactions when booting from NAND Flash in AutoRead mode, the NDFC gates the clocks to the EBC until the transactions complete on the NDFC interface. EBCC must be set to 1 for normal operation.

30	DHC	Direct Hardware Control Enable 0 Normal 1 Use NDFC0_HWCTL register to directly control and read external interface inputs and outputs	This bit enables the NDFC0_HWCTL register to control the NDFC external interface directly. The automatic control from the NDFC0_CMD, NDFC0_ADDR, NDFC0_DATA registers and the linear address region(s) are disabled when DHC is set. This should be used for debug purposes only.
31		Reserved	

36.9.8 NDFC Status Register (NDFC0_SR)

The NAND Flash Status Register (NDFC0_SR) is for monitoring the external Ready/Busy# indicator input that comes from the NAND Flash devices, indicating the currently active ECC accumulation register, and a status bit to indicate that read data loss may have occurred in accesses to an external NAND Flash device.

This register cannot be written. Unused bits return a value of 0.

Figure 36-18. NAND Flash Status Register (NDFC0_SR)

0		Reserved	
1:3	ECCP	Active NDFC0_ECC register identifier 0:7 = NDFC0_ECC0:NDFC0_ECC7	This field is reset to 000 when the NDFC0_CR[ERC] bit is written with a 1 to generate an ECC Reset. This field is automatically incremented after every 256-Bytes of data are transferred to the external interface. Once ECCP reaches 111 (NDFC0_ECC7) and 256 bytes of data are transferred, the field will not be incremented any further. This allows a full 2KB page to be transferred followed by its 64B spare data area without the spare data affecting the ECC value stored in any of the NDFC0_ECC registers.
4:5		Reserved	
6	RDL	Read Data Loss 0 No Read Data Loss has been detected 1 Possible Read data has been lost	This bit is only set when a EBC single byte-wide read access is performed to an external NAND Flash bank that is configured to be 16 bits wide, indicating that the upper 8 bits of read data from that device access were not returned. This may occur if byte read cycles are performed by the EBC to a 16-bit NAND Flash bank. This bit is only cleared when an CE# Reset via NDFC0_CR[CER] is performed. **Note:** This status bit is not cleared by changing the NDFC0_CR[BS] bank select.
7	RDY	Ready/Busy Input indicator 0 External NFRdyBusy input is low -> Busy 1 External NFRdyBusy input is high -> Ready	
8:31		Reserved	

36.9.9 NAND Flash Direct Hardware Control Register (NDFC0_HWCTL)

The NAND Flash Direct Hardware Control Register (NDFC0_HWCTL) enables direct control of the external interface inputs and outputs that go to the NAND Flash devices. The settings in this register have no control over the external interface unless the NDFC0_CR[DHC] bit is set.

This register is Read/Write with several conditions.

1. The contents of this register are not writable and reads will not return data in the DATAL and DATAH fields unless the NDFC0_CR[DHC] bit is already set.
2. The DATAL and DATAH fields are read-only when the DENL and DENH fields respectively are cleared (=0).

Unused bits return a value of 0

Figure 36-19. NAND Flash Direct Hardware Control Register (NDFC0_HWCTL)

0:3	CE0-CE3	Chip Enable $\overline{\text{NFCE0:3}}$	This 4-bit field contains bits CE0:CE3. The setting of these bits is inverted before going to the external output, so a setting of 1 here gives an active output (low) to the NAND Flash device's CE# input.
4	CLE	$\overline{\text{NFCLE}}$ direct control 0 $\overline{\text{NFCLE}}$ = 0 1 $\overline{\text{NFCLE}}$ = 1	The value of this bit directly correlates to the external output signal
5	ALE	NFALE direct control 0 $\overline{\text{NFALE}}$ = 0 1 $\overline{\text{NFALE}}$ = 1	The value of this bit directly correlates to the external output signal
6	RE	$\overline{\text{NFREn}}$ direct control (inverted) 0 $\overline{\text{NFREn}}$ = 1 1 $\overline{\text{NFREn}}$ = 0	The value of this bit is inverted before driving the external output signal.
7	WE	$\overline{\text{NFWEn}}$ direct control (inverted) 0 $\overline{\text{NFWEn}}$ = 1 1 $\overline{\text{NFWEn}}$ = 0	The value of this bit is inverted before driving the external output signal.
8	DENL	Direct control of External Device Write Data bus enable for NFData 00:07 0 PerData00:07 drivers are High-Z (Read) 1 PerData00:07 drivers are not High-Z (Write)	
9	DENH	0 Direct control of External Device Write Data bus enable for NFDATA08:15 0 PerData08:15 drivers are High-Z (Read) 1 PerData08:15 drivers are not High-Z (Write)	
10:15		Reserved	
16:23	DATAL	PerData00:07 direct control (DENL = 1) Read/Write PerData00:07 latched input value (DENL = 0) Read-only	When DENL is 0, this field returns the value on the PerData00:07 inputs. When DENL is 1, this field drives its value out to the PerData00:07 outputs.
24:31	DATAH	PerData08:15 direct control (DENH = 1) Read/Write PerData08:15 latched input value (DENH = 0) Read-only	When DENH is 0, this field returns the value on the PerData08:15 inputs. When DENH is 1, this field drives its value out to the PerData08:15 outputs.

36.9.10 NDFC Revision ID Register (NDFC0_REVID)

NDFC0_REVID is a read-only register that allows software to read the version information.

Figure 36-20. NAND Flash Revision ID Register (NDFC0_REVID)

0:11		Reserved	
12:23	RN	Revision Number	Corresponds to the RCS major revision number of the source RTL
24:31	BRN	Branch Revision Number	Corresponds to the RCS minor (branch) revision number of the source RTL

37. Universal Serial Bus (USB) 2.0 Host Controller (PPC460EX/EXr only)

The following sections provide detailed information for the Universal Serial Bus (USB) 2.0 Host controller implemented in the PPC460EX/EXr chip. USB is not available in the PPC460GT

37.1 USB Overview

The USB 2.0 Host Controller is fully compliant with the USB 2.0 specification, Enhanced Host Controller Interface (EHCI) Specification, Revision 1.0, and the Open Host Controller Interface (OHCI) Specification Release 1.0a. The controller supports high-speed, 480Mb/s transfers using an EHCI Host Controller, as well as full- and low speeds through an integrated OHCI Host Controller.

The USB 2.0 Host Controller includes the following features:

- General Features
  - All clock synchronization is handled within the controller.
  - Descriptor and data prefetching.
- Software Features
  - Complies with Enhanced Host Controller Interface (EHCI) Specification, Version 1.0, and the Open Host Controller Interface (OHCI) Specification, Version 1.0a.
- Application Features
  - AHB interface to the application.
  - Complies with the AMBA Specification, Revision 2.0.
  - Bus Interface Unit (BIU) handles retry, error and split transactions on the AHB.
  - As an AHB master, supports 8-, 16-, or 32-bit data transfers on the AHB.
  - Supports 32-bit addressing on the AHB.
- USB 2.0 Supported Features
  - Complies with the USB 2.0 Specification.
  - Supports ping and split transactions.
  - ULPI interface to the PHY.
  - ULPI PHY interface clock supports 60MHz operation
- Power Optimization Features
  - ULPI Reduced Power mode with ULPI wrapper operates at 60MHz, with the remainder of the Root Hub operating at 30MHz.

37.2 USB 2.0 Host Interface

The following sections describe the USB 2.0 Host interface.

The function consists of:

- One USB EHCI host controller that handles high-speed transfers.
- One USB OHCI host controller that handles full and low speed transfers.
- A root hub that selects the appropriate host controller to connect to the USB port.
- An AMBA AHB bus interface unit.

37.2.1 Functional Description

The USB 2.0 Host controller is designed with an AHB interface at the application side and a Universal Transceiver Macrocell Interface (UTMI) at the transceiver side. The USB 2.0 Host controller contains one OHCI Host Controller which handles full/low speed transfers (12Mbps/1.5Mbps) and one EHCI Host Controller which handles high speed transfers (480Mbps). The AHB provides a connection to the rest of the system through the PLB-to-AHB Bridge.

The AHB slave interface is used by the Host Controller Driver (HCD) to access the EHCI Host Controller capability, operational, and implementation specific registers and the OHCI host controller operational registers. The AHB master interface is used by the OHCI and EHCI Host Controllers to access the USB data structures and data buffers defined by the HCD.

In the case of EHCI, when the Run/Stop bit is set to 1, the EHCI Host Controller begins to do list processing by fetching the descriptors from the system memory through the AHB master interface. The EHCI Host Controller is capable of prefetching descriptors and data and supports a fixed number of four prefetched descriptors (both periodic and non-periodic). For prefetching data it uses an adaptive buffering scheme that allocates and deallocates buffers dynamically. Total buffer size is fixed at 4 KB. This buffer is divided into eight 512B segments. When the EHCI Host Controller prefetches the descriptor, it also prefetches data for OUT endpoints. For example, if there are four OUT transactions to be performed, while the data is being transferred on the USB for the first OUT transaction, the remaining three OUT descriptors and data can be prefetched. This overlapped operation boosts performance.

The EHCI Host Controller generates and sends an appropriate token to the USB device, based on the type of request (ISO, bulk, interrupt or control). For OUT transactions, prefetched data is read from system memory and sent out to the USB device. For IN transactions, the data read from the USB device is written to system memory. At the end of the transfer, the transaction status is updated in the descriptor in system memory. This descriptor status update repeats in every microframe. If there is a memory error during any memory transaction, the System Error bit is set in the Command register, the Run/Stop bit is cleared and the EHCI Host Controller moves to the Halted state.

The EHCI Host Controller performs all front-end functions, such as token packet decoding, CRC checking and generation, and byte packing and unpacking, before it transmits this data to or receives it from the external ULPI PHY.

The USB 2.0 Host controller's connection to the USB wire is provided through a one-port ULPI PHY transceiver interface. This interface is connected to a ULPI-compliant PHY device.

37.2.2 Power Management

Power management for the USB function operates as described in the following sections.

37.2.2.1 USB EHCI Host Controller

The EHCI Host Controller provides power management capabilities under control of system software. The first step is to suspend the USB port if it is enabled. This halts the USB2HClk from the PHY. The next step is to shut off the host controller by setting the Run/Stop bit in the USBCMD register to a zero. The application software must stop all clocks and activities on the bus to place the EHCI Host Controller in low-power mode.

When a wake event occurs the system resumes operation and system software eventually sets the Run/Stop bit to a one and resumes the suspended ports. Software must not set the Run/Stop bit to a one until it is confirmed that the clock to the host controller is stable.

37.2.2.2 USB OHCI Host Controller

The OHCI Host Controller does not provide power management capabilities to switch the 12-MHz and 48-MHz clocks on/off for power savings.

37.2.3 User Model for HCRESET During ULPI, Exiting Low-power Mode

It is strongly recommended that before issuing HCRESET, all ports should be taken out of low-power mode and resume normal operation. When this recommendation is not followed, a host meeting specific requirements can use an alternate procedure for issuing HCRESET. The host requirements and procedure follow:

The Host must meet these requirements:
- ULPI configuration is required.
- The port must be owned by EHCI.
- Port power switching capability must be present.

Procedure:

1. Set the HCReset bit in the USB Command register (USBCMD) from the AHB side.

   This operation has the following effects:
   - Resets the running portion of the host.
   - Turns port power off.
   - Deasserts suspend internally.

Together they cause the PHY to restart USB2HClk.

2. Wait 1 ms for USB2HClk to begin running.
3. Set PORTSC.PP bit to 1. This operation turns port power on.
4. Issue HCReset again as in Step 1.

   This resets the entire host, including the ports.

37.2.4 ULPI Interface Operation

The ULPI interface connects to an external high-speed PHY.

Table 37-1. ULPI Interface Signals

Signal Name	I/O	Description
USB2HClk	I	ULPI clock. Receives the 60 MHz clock supplied by the high-speed ULPI PHY.
USB2HDir	I	Data bus direction control. When the PHY has data to transfer to the USB Host controller, it drives USB2HDir high to take ownership of the bus. When the PHY has no data to transfer, it drives USB2HDir low and monitors the bus. 0 USB Host is the driver 1 ULPI PHY is the driver
USB2HNext	I	Next data control. When the USB Host controller function is sending data to the PHY, USB2HNext indicates when the current byte is accepted by the PHY. The USB Host controller function places the next byte on the data bus in the following clock cycle. When the PHY is sending data to the USB Host controller function, USB2HNext indicates when a new byte is available for the USB Host controller function to consume.

Table 37-1. ULPI Interface Signals (Continued)

Signal Name	I/O	Description
USB2HD0:7	I/O	ULPI data I/O. ULPI data input to the function driven by the PHY. ULPI data output bus driven by the USB Host controller function.
USB2HStop	O	Stop output control. The USB Host controller function asserts this signal for one clock cycle to indicate the end of a USB transmit packet or a register write operation and, optionally, to stop packet reception. Assertion occurs after the last data byte is presented on the bus.

37.2.5 AHB USB Interface

See *AHB Subsystem SDR Registers* on page 136. Details for the SDR0_USB2HOST_CFG and SDR0_USB2HOST_STS registers are provided at that location.

37.2.6 USB 2.0 Host Controller Registers

The USB 2.0 Host controller contains two independent sets of registers:
- EHCI host controller Capability, Operational and implementation specific registers
- OHCI host controller Operational registers

Table 37-2 lists the registers provided for the USB 2.0 Host controller. The EHCI and OHCI registers are memory mapped. The following sections define the bit-level detail for the registers.

Table 37-2. USB 2.0 Host Controller Registers

Mnemonic	Register	Address	Access	Page
EHCI Registers				
EHCI0_HCCAPBASE	Version/Capability Length	0x4 BFFD 0400	R	1294
EHCI0_HCSPARAMS	Structural Parameters	0x4 BFFD 0404	R	1294
EHCI0_HCCPARAMS	Capability Parameters	0x4 BFFD 0408	R	1296
EHCI0_USBCMD	USB Command	0x4 BFFD 0410	See register	1297
EHCI0_USBSTS	USB Status	0x4 BFFD 0414	R/W	1298
EHCI0_USBINTR	USB Interrupt Enable	0x4 BFFD 0418	R/W	1301
EHCI0_FRINDEX	EHCI Frame Index	0x4 BFFD 041C	R/W	1302
EHCI0_CRTLSEG	Control Data Structure Segment	0x4 BFFD 0420	R/W	1303
EHCI0_PRDLISTB	Periodic Frame List Base Address	0x4 BFFD 0424	R/W	1303
EHCI0_ASYNCLSTA	Current Asynchronous List Address	0x4 BFFD 0428	R/W	1304
EHCI0_CFGFLAG	Configure Flag	0x4 BFFD 0450	R/W	1304
EHCI0_PORTSC	Port Status and Control	0x4 BFFD 0454	R/W	1304
EHCI0_INSNREG0	Programmable Microframe Base Value	0x4 BFFD 0490	R/W	1309
EHCI0_INSNREG4	Debug Only	0x4 BFFD 04A0	R/W	1309
EHCI0_INSNREG5	ULPI Configuration Register	0x4 BFFD 04A4	R/W	1310

Table 37-2. USB 2.0 Host Controller Registers (Continued)

Mnemonic	Register	Address	Access	Page
OHCI Registers				
OHCI0_HCREV	Revision	0x4 BFFD 0000	R	1310
OHCI0_HCCTRL	Control	0x4 BFFD 0004	R/W	1311
OHCI0_HCCMDSTS	Command Status	0x4 BFFD 0008	See register	1312
OHCI0_HCINTSTS	Interrupt Status	0x4 BFFD 000C	R/W	1314
OHCI0_HCINTE	Interrupt Enable	0x4 BFFD 0010	R/W	1315
OHCI0_HCINTDIS	Interrupt Disable	0x4 BFFD 0014	R/W	1316
OHCI0_HCHCCA	Host Controller Communications Area	0x4 BFFD 0018	R/W	1317
OHCI0_HCPCED	Periodic Current Endpoint Descriptor	0x4 BFFD 001C	R/W	1317
OHCI0_HCCHED	Control Head Endpoint Descriptor	0x4 BFFD 0020	R/W	1318
OHCI0_HCCTRLCED	Control Current Endpoint Descriptor	0x4 BFFD 0024	R/W	1318
OHCI0_HCBULKHED	Bulk Head Endpoint Descriptor	0x4 BFFD 0028	R/W	1319
OHCI0_HCBULKCED	Bulk Current Endpoint Descriptor	0x4 BFFD 002C	R/W	1319
OHCI0_HCDHEAD	Done Head Transfer Descriptor	0x4 BFFD 0030	R	1320
OHCI0_HCFMINT	Frame Interval	0x4 BFFD 0034	R/W	1321
OHCI0_HCFMREM	Frame Remaining	0x4 BFFD 0038	R	1322
OHCI0_HCFMNUM	Frame Number	0x4 BFFD 003C	R	1322
OHCI0_HCPRDSTRT	Periodic Start	0x4 BFFD 0040	R/W	1323
OHCI0_HCLSTHRES	Low Speed Threshold	0x4 BFFD 0044	R/W	1323
OHCI0_HCRHDESCA	Root Hub Descriptor A	0x4 BFFD 0048	See register	1324
OHCI0_HCRHDESCB	Root Hub Descriptor B	0x4 BFFD 004C	R/W	1325
OHCI0_HCRHSTS	Root Hub Status	0x4 BFFD 0050	See register	1326
OHCI0_HCPRSTS	Root Hub Port Status	0x4 BFFD 0054	R/W	1327

37.2.6.1 Version/Capability Length Register (EHCI0_HCCAPBASE)

This register contains the 2-byte interface version number and a 1-byte capability register length.

Reset: 0x0100_0010

Figure 37-1. Version/Capability Length Register (EHCI0_HCCAPBASE)

Bit	Mnemonic	Description	Comments
0:15	HCIVER	Interface version number EHCI revision number supported by this Host controller. The most significant byte of this register represents a major revision and the least significant byte is the minor revision.	
16:11		Reserved	
24:31	CAPLTH	Capability register length Offset to add to register base to find the beginning of the Operational Register Space	

37.2.6.2 Structural Parameters Register (EHCI0_HCSPARAMS)

This register contains the structural parameters, such as the number of downstream ports, port indicators, and so on.

Reset: 0x0000_1111

Figure 37-2. Structural Parameters Register (EHCI0_HCSPARAMS)

Bit	Mnemonic	Description	Comments
0:7		Reserved	
8:11	DPN	Debug Port Number Identifies which of the host controller ports is the debug port. The value is the port number (one-based) of the debug port. A non-zero value in this field indicates the presence of a debug port.	The value in this register must not be greater than EHCI0_HCSPARAMS[NPORTS].
12:14		Reserved	
15	PIND	Port Indicators Indicates whether the ports support port indicator control. When this bit is 1, the port status and control registers include a R/W field for controlling the state of the port indicator.	
16:19	NCC	Number of Companion Controllers Number of companion controllers associated with this USB 2.0 host controller. 0000 No companion host controllers. Port-ownership hand-off is not supported. Only high-speed devices are supported on the host controller root ports. 0001 A non-zero value in this field indicates there are companion USB 1.1 host controller(s). Port-ownership hand-offs are supported. High, full- and low-speed devices are supported on the host controller root ports.	
20:23	NPCC	Number of Ports per Companion Controller Number of ports supported per companion host controller.	Used to indicate the port routing configuration to system software. The value in this field must be consistent with EHCI0_HCSPARAMS[NPORTS] and EHCI0_HCSPARAMS[NCC].
24	PRR	Port Routing Rules Indicates the method used by this implementation for mapped of ports to companion controllers. 0 First NPCC port is routed to the lowest numbered function companion host controller, the next NPCC port is routed to the next lowest function companion controller, and so on. 1 Port routing is explicitly enumerated by the first NPORTS elements of the HCSP-PORTROUTE array.	
25:26		Reserved	
27	PPC	Port Power Control Indicates whether the host controller implementation includes port power control. 0 Ports do not have port power switches 1 Ports have port power switches	The value of this field affects the functionality of the Port Power field in each port status and control register.
28:31	NPORTS	Number of Downstream Ports Specifies the number of physical downstream ports implemented on this host controller.	The value of this field determines how many port registers are addressable in the Operational Register Space. Valid values are in the range of 0x1 to 0xF. A zero in this field is undefined.

37.2.6.3 Capability Parameters Register (EHCI0_HCCPARAMS)

This register provides USB Multiple Mode control (time-base bit functionality), addressing capability.

Reset: 0x0000_0026

Figure 37-3. Capability Parameters Register (EHCI0_HCCPARAMS)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:23	EECP	EHCI Extended Capabilities Pointer Indicates the existence of a capabilities list. A value of zero indicated no extended capabilities are implemented. A non-zero value indicates the offset in configuration space of the first EHCI extended capability.	Default is implementation dependent. The pointer value must be 0x40 or greater if implemented to maintain the consistency of the PCI header defined for this class of device.
24:27	IST	Isochronous Scheduling Threshold Indicates, relative to the current position of the executing host controller, where software can reliably update the isochronous schedule. When bit 24 is 0, the value of the least significant 3 bits indicates the number of micro-frames a host controller can hold a set of isochronous data structures (one or more) before flushing the state. When bit 24 is 1, host software assumes the host controller may cache an isochronous data structure for an entire frame.	Default is implementation dependent.
28		Reserved	
29	ASPC	Asynchronous Schedule Park Capability 0 Park feature is not supported 1 Park feature for high-speed queue heads in the Asynchronous Schedule is supported.	Default is implementation dependent See EHCI0_USBCMD[ASPME] and EHCI0_USBCMD[ASPMC].
30	PFLF	Programmable Frame List Flag 0 System software must use a frame list length of 1024 elements 1 System software can specify and use a smaller frame list and configure the Host Controller by means of the EHCI0_USBCMD register Frame List Size field.	Default is implementation dependent. The frame list must always be aligned on a 4K page boundary. This ensures that the frame list is always physically contiguous.
31	64BAC	64-bit Addressing Capability Addressing range capability 0 32-bit address memory pointers 1 64-bit address memory pointers	This is not tightly coupled with the USBBASE address register mapping control. Sixty-four bit Addressing Capability indicates the host controller can generate 64-bit addresses as a master. The USBBASE register indicates the host controller only needs to decode 32-bit addresses as a slave.

37.2.6.4 USB Command Register (EHCI0_USBCMD)

This register indicates the command to be executed by the USB Host Controller. Writing to this register causes a command to be executed.

Reset: 0x0008_0B00

Figure 37-4. USB Command Register (EHCI0_USBCMD)

Bit	Mnemonic	Description	Comments
0:7		Reserved	
8:15	ITC	Interrupt Threshold Control Select the maximum rate at which the host controller will issue interrupts. The only valid values are defined below. If software writes an invalid value to this register, the results are undefined. 0x00 Reserved 0x01 1 micro-frame 0x02 2 micro-frames 0x04 4 micro-frames 0x08 8 micro-frames (default, equates to 1ms) 0x10 16 micro-frames (2ms) 0x20 32 micro-frames (4ms) 0x40 64 micro-frames (8ms)	Software modifications to this bit while EHCI0_USBSTS[HCH] = 0 results in undefined behavior.
16:19		Reserved	
20	ASPME	Asynchronous Schedule Park Mode Enable 0 Disable Park Mode 1 Enable Park Mode	If EHCI0_HCCPARAMS[ASPC] = 1, ASPME defaults to 1 and is R/W. Otherwise, ASPME must be 0 and is Read only.
21		Reserved	
22:23	ASPMC	Asynchronous Schedule Park Mode Count Contains a count of the number of successive transactions the host controller is allowed to execute from a high-speed queue head on the Asynchronous schedule before continuing traversal of the Asynchronous schedule. Valid values are 0x1 to 0x3.	If EHCI0_HCCPARAMS[APMC] = 1, this field defaults to 0x3 and is R/W. Otherwise, it defaults to 0 and is Read only. Software must not write 0 to this bit when ASPME = 1, as this results in undefined behavior.
24	LHRC	Light Host Controller Reset 0 Light Host Controller Reset has completed and it is safe for host software to re-initialize the host controller 1 Light Host Controller Reset has not yet completed	This control bit is not required. If implemented, it allows the driver to reset the EHCI controller without affecting the state of the ports or the relationship to the companion host controllers. If not implemented, a read of this field always returns a zero.
25	IAAD	Interrupt on Asynchronous Advance Doorbell 0 Host Controller should not issue an interrupt the next time it advances the asynchronous schedule 1 Host Controller should issue an interrupt the next time it advances the asynchronous schedule	When the Host Controller has evicted all appropriate cached schedule states, it sets EHCI0_USBSTS[IAA] = 1. If EHCI0_USBSTS[IAAE] = 1 the Host Controller asserts an interrupt at the next interrupt threshold. The host controller sets this bit to 0 after it sets EHCI0_USBSTS[IAA] = 1.Software should not write a one to this bit when the asynchronous schedule is disabled. Doing so will yield undefined results.

Figure 37-4. USB Command Register (EHCI0_USBCMD) (Continued)

Bit	Mnemonic	Description	Comments
26	ASE	Asynchronous Schedule Enable 0 Do not process the Asynchronous Schedule 1 Process the Asynchronous Schedule	Default = 0 Use the ASYNCLISTADDR register to access the Asynchronous Schedule.
27	PSE	Periodic Schedule Enable 0 Do not process the Periodic Schedule 1 Process the Periodic Schedule	Default = 0 Use the PERIODICLISTBASE register to access the Periodic Schedule.
28:29	FLS	Frame List Size Specifies the size of the frame list. The size the frame list controls which bits in the Frame Index Register should be used for the Frame List Current index. 00 1024 elements (4096 bytes) 01 512 elements (2048 bytes) 10 256 elements (1024 bytes) 11 Reserved	Default = 00 This field is R/W only if EHCI0_HCCPARAMS[PFLF] =1.
30	HCRST	Host Controller Reset 0 Host Controller reset process is complete 1 Host Controller resets its internal pipelines, timers, counters, state machines, etc. to their initial value. Any transaction currently in progress on the USB is immediately terminated.	A USB reset is not driven on downstream ports. All operational registers, including port registers and port state machines are set to their initial values. Port ownership reverts to the companion host controller(s). Software must reinitialize the host controller in order to return the host controller to an operational state. Software cannot terminate the reset process early by writing 0 to this bit. Software should not set this bit to 1 when EHCI0_USBSTS[HCH] = 0. Attempting to reset an actively running Host Controller results in undefined behavior.
31	RS	Run/Stop 0 Stop. Host Controller completes the current and any actively pipelined transactions on the USB and then halts. 1 Run. Host Controller proceeds with execution of the schedule. The Host Controller continues execution as long as this bit is set to a 1.	Default = 0 The Host Controller must halt within 16 micro-frames after software clears the RS bit. EHCI0_USBSTS[HCH] indicates when the Host Controller has finished its pending pipelined transactions and has entered the stopped state. Software must not write a one to this field unless EHCI0_USBSTS[HCH] = 1. Doing so produces undefined results.

37.2.6.5 USB Status Register (EHCI0_USBSTS)

This register indicates pending interrupts and various states of the Host Controller. The status resulting from a transaction on the USB is not indicated in this register. Software sets a bit to 0 in this register by writing a 1 to it.

Access: See field specification

Reset: 0x0000_1000

Figure 37-5. USB Status Register (EHCI0_USBSTS)

Bit	Mnemonic	Description	Comments
0:15		Reserved	

Figure 37-5. USB Status Register (EHCI0_USBSTS) (Continued)

Bit	Mnemonic	Description	Comments
16	ASS	Asynchronous Schedule Status Reports the current real status of the Asynchronous Schedule. 0 Disable status of the Asynchronous Schedule 1 Enable status of the Asynchronous Schedule	Default = 0 Read only The Host Controller is not required to immediately disable or enable the Asynchronous Schedule when software changes EHCI0_USBCMD[ASE]. When this bit and EHCI0_USBCMD[ASE] are the same value, the Asynchronous Schedule is either enabled (1) or disabled (0).
17	PSS	Periodic Schedule Status Reports the current real status of the Periodic Schedule. 0 Disable status of the Periodic Schedule 1 Enable status of the Periodic Schedule	Default = 0 Read only The Host Controller is not required to immediately disable or enable the Periodic Schedule when software changes EHCI0_USBCMD[PSE]. When this bit and EHCI0_USBCMD[PSE] are the same value, the Periodic Schedule is either enabled (1) or disabled (0).
18	R	Reclamation Detects an empty asynchronous schedule.	Default = 0 Read only
19	HCH	Host Controller Halted 0 Run/Stop bit = 1. 1 Host Controller has stopped executing as a result of the Run/Stop bit set to 0 by either software or the Host Controller hardware (for example, internal error).	Default = 1 Read only
20:25		Reserved	
26	IAA	Interrupt on Asynchronous Advance Status of EHCI0_USBCMD[IAAD].	Default = 0 R/W Clear
27	HSE	Host System Error 0 No error during a host system access 1 Error during a host system access	When this error occurs, the Host Controller clears EHCI0_USBCMD[RS] to prevent further execution of the scheduled Transfer Descriptors (TDs).
28	FLR	Frame List Rollover 0 Frame List Index has not rolled over from its maximum value 1 Frame List Index has rolled over from its maximum value to 0	R/W Clear The exact value at which rollover occurs depends on the frame list size. For example, if the frame list size (as programmed in EHCI0_USBCMD[FLS]) is 1024, the Frame Index Register rolls over every time EHCI0_FRINDEX[18] toggles. Similarly, if the size is 512, the Host Controller sets this bit to a one every time EHCI0_FRINDEX[17] toggles.
29	PCD	Port Change Detect The Host Controller sets this bit to 1 when: Any port for which the Port Owner bit is set to 0 changes from a 0 to 1 or a Force Port Resume bit changes from 0 to 1 as a result of a J-K transition detected on a suspended port. or As a result of the Connect Status Change being set to 1 after system software has relinquished ownership of a connected port by writing a 1 to a port's Port Owner bit.	R/W Clear This bit is allowed to be maintained in the Auxiliary power well. Alternatively, it is also acceptable that on a D3 to D0 transition of the EHCI HC device, this bit is loaded with the OR of all of the PORTSC change bits (including: Force port resume, over-current change, enable/disable change and connect status change).

Figure 37-5. USB Status Register (EHCI0_USBSTS) (Continued)

Bit	Mnemonic	Description	Comments
30	ERRINT	USB Error Interrupt The Host Controller sets this bit 1 when completion of a USB transaction results in an error condition (for example, error counter underflow). If the TD on which the error interrupt occurred also had its IOC bit set, both ERRINT and USBINT bit are set.	R/W Clear
31	USBINT	USB Interrupt The Host Controller sets this bit to 1: On the completion of a USB transaction, which results in the retirement of a TD that had its IOC bit set. or When a short packet is detected.	R/W Clear

37.2.6.6 USB Interrupt Enable Register (EHCI0_USBINTR)

This register enables and disables reporting of the corresponding interrupt to the software. When a bit is set and the corresponding interrupt is active, an interrupt is generated to the host. Interrupt sources that are disabled in this register still appear in the EHCI0_USBSTS register to allow the software to poll for events.

Each interrupt enable bit description indicates whether it is dependent on the interrupt threshold mechanism.

Reset: 0x0000_0000

Figure 37-6. USB Interrupt Enable Register (EHCI0_USBINTR)

Bit	Mnemonic	Description	Comments
0:26		Reserved	
26	IAAE	Interrupt on Asynchronous Advance Enable When this bit = 1 and EHCI0_USBSTS[IAA] = 1, the Host Controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by clearing EHCI0_USBSTS[IAA].	
27	HSEE	Host System Error Enable When this bit = 1 and EHCI0_USBSTS[HSE] = 1, the Host Controller issues an interrupt. The interrupt is acknowledged by clearing EHCI0_USBSTS[HSE].	
28	FLRE	Frame List Rollover Enable When this bit = 1 and EHCI0_USBSTS[FLR] = 1, the Host Controller issues an interrupt. The interrupt is acknowledged by clearing EHCI0_USBSTS[FLR].	
29	PCDE	Port Change Detect Enable When this bit = 1 and EHCI0_USBSTS[PCD] = 1, the Host Controller issues an interrupt. The interrupt is acknowledged by clearing EHCI0_USBSTS[PCD].	
30	USBEIE	USB Error Interrupt Enable When this bit = 1 and EHCI0_USBSTS[ERRINT] = 1, the Host Controller issues an interrupt. The interrupt is acknowledged by clearing EHCI0_USBSTS[ERRINT].	
31	USBIE	USB Interrupt Enable When this bit = 1 and EHCI0_USBSTS[USBINT] = 1, the Host Controller issues an interrupt. The interrupt is acknowledged by clearing EHCI0_USBSTS[USBINT]	

37.2.6.7 EHCI Frame Index Register (EHCI0_FRINDEX)

This register is used by the Host Controller to index into the periodic frame list. The register updates every 125 μs (once each micro-frame). Bits N:3 are used to select a particular entry in the Periodic Frame List during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in EHCI0_USBCMD[FLS].

This register must be written as a double word. Byte writes produce undefined results. This register cannot be written unless the Host Controller is in the Halted state as indicated by EHCI0_USBSTS[HCH]. A write to this register while EHCI0_USBSTS[RS] = 1 produces undefined results. Writes to this register also affect the SOF value.

The SOF frame number value for the bus SOF token is derived or alternatively managed from this register. The value of FRINDEX must be 125 μs (1 micro-frame) ahead of the SOF token value. The SOF value can be implemented as an 11-bit shadow register. For this discussion, this shadow register is 11 bits and is named SOFV. SOFV updates every 8 micro-frames. (1 ms). An example implementation to achieve this behavior is to increment SOFV each time the FRINDEX[2:0] increments from 0 to 1.

Software must use the value of FRINDEX to derive the current micro-frame number, both for high-speed isochronous scheduling purposes and to provide the get micro-frame number function required for client drivers. Therefore, the value of FRINDEX and the value of SOFV must be kept consistent if the chip is reset or software writes to FRINDEX. Writes to FRINDEX must also write-through FRINDEX[13:3] to SOFV[10:0]. In order to keep the update as simple as possible, software should never write a FRINDEX value where the three least significant bits are 0b111 or 0b000.

Reset: 0x0000_0000

Figure 37-7. EHCI Frame Index Register (EHCI0_FRINDEX)

Bit	Mnemonic	Description	Comments
0:17		Reserved	
18:31	FI	Frame Index The value in this register increments at the end of each time frame (for example, a micro-frame). Bits N:3 are used for the Frame List current index. This means that each location of the frame list is accessed eight times (frames or micro-frames) before moving to the next index. The following illustrates values of N based on the value of the Frame List Size field in the USBCMD register: USBCMD[FLS] Number Elements N 00 1024 12 01 512 11 10 256 10 11 Reserved	

37.2.6.8 Control Data Structure Segment Register (EHCI0_CRTLSEG)

This register corresponds to the most significant address bits 63:32 for all EHCI data structures. If EHCI0_HCCPARAMS[64BAC] = 0, this register is not used. Software cannot write to it, and a read from this register returns zeros.

If the EHCI0_HCCPARAMS[64BAC] = 1, this register is used with the link pointers to construct 64-bit addresses to EHCI control data structures. This register is concatenated with the link pointer from either the PERIODICLISTBASE, ASYNCLISTADDR, or any control data structure link field to construct a 64-bit address.

This register allows software to locate all control data structures within the same 4 GB memory segment.

Reset: 0x0000_0000

Figure 37-8. Control Data Structure Segment Register (EHCI0_CRTLSEG)

Bit	Mnemonic	Description	Comments
0:31	4GCDA	4 GB Control Data Structure Segment Address	

37.2.6.9 Periodic Frame List Base Address Register (EHCI0_PRDLISTB)

This 32-bit register contains the beginning address of the Periodic Frame List in the system memory. If the Host Controller is in 64-bit mode (EHCI0_HCCPARAMS[64BAC] = 1), the most significant 32 bits of every control data structure address comes from EHCI0_CRTLSEG. System software loads this register prior to starting the schedule execution by the Host Controller. The memory structure referenced by this physical memory pointer is assumed to be 4-KB aligned. The contents of this register are combined with EHCI0_FRINDEX to enable the Host Controller to step through the Periodic Frame List in sequence.

Reset: 0x0000_0000

Figure 37-9. Periodic Frame List Base Address Register (EHCI0_PRDLISTB)

Bit	Mnemonic	Description	Comments
0:19	LOW	Base Address Corresponds to memory address signals 0:19, respectively.	
20:31		Reserved	

37.2.6.10 Current Asynchronous List Address Register (EHCI0_ASYNCLSTA)

This register contains the address of the next asynchronous queue head to be executed. If the Host Controller is in 64-bit mode (EHCI0_HCCPARAMS[64BAC] = 1), the most significant 32 bits of every control data structure address comes from EHCI0_CRTLSEG. The memory structure referenced by this physical memory pointer is assumed to be 32-byte (cache line) aligned.

Reset: 0x0000_0000

Figure 37-10. Current Asynchronous List Address Register (EHCI0_ASYNCLSTA)

Bit	Mnemonic	Description	Comments
0:26	LPL	Link Pointer Low Corresponds to memory address signals 0:26, respectively.	This field may only reference a Queue Head (QH).
27:31		Reserved	

37.2.6.11 Configure Flag Register (EHCI0_CFGFLAG)

This register is in the auxiliary power well. It is only reset by hardware when the auxiliary power is initially applied or in response to a Host Controller reset.

Reset: 0x0000_0000

Figure 37-11. Configure Flag Register (EHCI0_CFGFLAG)

Bit	Mnemonic	Description	Comments
0:30		Reserved	
31	CF	Configure Flag Controls the default port-routing control logic. 0 Port routing control logic default-routes each port to an implementation dependent classic Host Controller. 1 Port routing control logic default-routes all ports to this Host Controller.	Host software sets this bit as the last action in its process of configuring the Host Controller.

37.2.6.12 Port Status and Control Register (EHCI0_PORTSC)

A Host Controller must implement one or more port registers. The number of port registers implemented by a particular instantiation of a host controller is documented in EHCI0_HCCPARAMS. Software uses this information as an input parameter to determine how many ports need to be serviced. All ports have the structure defined below.

This register is in the auxiliary power well. It is only reset by hardware when the auxiliary power is initially applied or in response to a host controller reset. The initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, software cannot change the state of the port until after it applies power to the port by setting port power to a 1. Software must not attempt to change the state of the port until after power is stable on the port. The host is required to have power stable to the port within 20ms of the 0 to 1 transition.

Notes:

1. When a device is attached, the port state changes to the connected state and system software processes this as with any status change notification.
2. If a port is being used as the Debug Port, then the port may report device connected and enabled when the Configured Flag is 0.

Access: See field specification

Reset: 0x0000_2000

Table 37-3. Port Status and Control Register (EHCI0_PORTSC)

Bit	Mnemonic	Description	Comments
0:8		Reserved	
9	WKOCE	Wake on Over-current Enable Writing this bit to a 0 Port Power is 0 1 Enables the port for sensitivity to over-current conditions as wake-up events	R/W
10	WKDCTE	Wake on Disconnect Enable 0 Port Power is 0 1 Enables the port for sensitivity to device disconnects as wake-up events	R/W
11	WKCTE	Wake on Connect Enable 0 Port Power is 0 1 Enables the port for sensitivity to device connects as wake-up events	R/W
12:15	PTC	Port Test Control 0000 Port is *not* operating in a test mode 0001 Test J state 0010 Test K state 0011 Test SE0_NAK 0100 Test packet 0101 Test FORCE_ENABLE 0110 ... 1111 Reserved	R/W
16:17	PIC	Port Indicator Control If EHCI0_HCCPARAMS[PIND] is 1, then: 00 Port Power is 0 or port indicators are off 01 Amber 10 Green 11 Undefined	R/W
18	PO	Port Owner 0 EHCI0_CFGFLAG[CF] made a 0 to 1 transition 1 EHCI0_CFGFLAG[CF] is 0	R/W System software uses this field to release ownership of the port to a selected Host Controller (in the event that the attached device is not a high-speed device). Software writes 1 to this bit when the attached device is not a high-speed device.

Table 37-3. Port Status and Control Register (EHCI0_PORTSC) (Continued)

Bit	Mnemonic	Description	Comments
19	PP	Port Power The function of this bit depends on the value of EHCI0_HCCPARAMS[PPC] 0 Port power control switch is off 1 Port power control switch is on	R/W This bit is Read only if the Host Controller does not have port power control switches. Each port is hard-wired to power. When power is not available on a port (that is, PP = 0), the port is nonfunctional and does not report attaches, detaches, and so on. When an over-current condition is detected on a powered port and PPC is 1, the PP bit in each affected port can be changed by the Host Controller from a 1 to 0 (removes power from the port).
20:21	LS	Line Status These bits are used for detection of low-speed USB devices prior to the port reset and enable sequence. USB State Interpretation 00 SE0 Not low-speed device, perform EHCI reset 01 J-state Not low-speed device, perform EHCI reset 10 K-state Low-speed device, release ownership of port 11 Undefined Not Low-speed device, perform EHCI reset.	Read only Undefined if EHCI0_PORTSC[PP] = 0. This field is valid only when the port enable bit is 0 and the current connect status bit is 1.
22		Reserved	
23	PR	Port Reset 0 Port is not in reset 1 Port is in reset	R/W When software changes this bit from 0 to1, the bus reset sequence as defined in the USB Specification Revision 2.0 is started. Software writes 0 to this bit to terminate the bus reset sequence. Software must keep this bit at 1 long enough to ensure the reset sequence completes. **Note:** When software writes 1 to this bit, it must also write 0 to the Port Enable bit. When software writes 0 to this bit there might be a delay before the bit status changes to a 0. The bit status will not read as a 0 until after the reset has completed. If the port is in high-speed mode after reset is complete, the host controller will automatically enable this port (that is, set the Port Enable bit to 1). A Host Controller must terminate the reset and stabilize the state of the port within 2 ms of software changing this bit from 1 to 0. For example, if the port detects that the attached device is high-speed during reset, the Host Controller must have the port in the enabled state within 2ms of software setting this bit to 0. EHCI0_USBSTS[HCH] should be 0 before software attempts to use this bit. The Host Controller may hold Port Reset asserted to 1 when EHCI0_USBSTS[HCH] = 1. This field is 0 if Port Power is 0.

Table 37-3. Port Status and Control Register (EHCI0_PORTSC) (Continued)

Bit	Mnemonic	Description	Comments
24	S	Suspend 0 Port not in suspended state 1 Port in suspended state	R/W The Port Enabled and Suspend bits of this register define the port states as follows: Bits Port State 0x Disable 10 Enable 11 Suspend When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction, if a transaction was in progress when this bit was written to 1. In the suspend state, the port is sensitive to resume detection. Note that the bit status does not change until the port is suspended and that there may be a delay in suspending a port if there is a transaction currently in progress on the USB. A write of zero to this bit is ignored by the host controller. The host controller will unconditionally set this bit to a zero when: • Software changes the Force Port Resume bit from 1 to 0. • Software changes the Port Reset bit from 0 to1. If Host software sets this bit to a one when the port is not enabled (that is, Port Enabled = 0) the results are undefined. This field is 0 if Port Power is 0.
25	FPR	Force Port Resume 0 No resume (K-state) detected/driven on port 1 Resume detected/driven on port	R/W The functionality defined for manipulating this bit depends on the value of the Suspend bit. For example, if the port is not suspended (Suspend and Enabled bits are a one) and software changes this bit to 1, the effects on the bus are undefined. Software sets this bit to 1 to drive resume signaling. The Host Controller sets this bit to 1 if a J-to-K transition is detected while the port is in the Suspend state. When this bit changes to 1 because a J-to-K transition is detected, the Port Change Detect bit in the USBSTS register is set to 1. If software sets this bit to 1, the hOst Controller must not set the Port Change Detect bit. When the EHCI controller owns the port, the resume sequence follows the sequence documented in the USB Specification Revision 2.0. The resume signaling (Full-speed 'K') is driven on the port as long as this bit remains a one. Software must appropriately time the Resume and set this bit to 0 when the appropriate amount of time has elapsed. Changing the bit from 0 to 1 causes the port to return to high-speed mode (forcing the bus below the port into a high-speed idle). This bit remains 1 until the port switches to the high-speed idle. The host controller must complete this transition within 2ms of software setting this bit to 0. This field is zero if Port Power is zero.
26	OCC	Over Current Change 0 No change to over current active 1 Change to over current active occurred.	Read with Clear Software clears this bit by writing 1.

Table 37-3. Port Status and Control Register (EHCI0_PORTSC) (Continued)

Bit	Mnemonic	Description	Comments
27	OCA	Over Current Active 0 No over current condition on this port 1 Over current condition exists on this port	Read only This bit automatically clears when the over current condition is removed.
28	PEDC	Port Enable/Disable Change 0 No change 1 Port enabled/disabled status has changed	Read with Clear For the root hub, this bit is 1 only when a port is disabled due to the appropriate conditions existing at the EOF2 point. Software clears this bit by writing 1. This field is 0 if Port Power is 0.
29	PED	Port Enable/Disable 0 Disable 1 Enable	Ports can only be enabled by the Host Controller as a part of the reset and enable. Software cannot enable a port by writing 1 to this field. The host controller sets this bit to 1 only when the reset sequence determines that the attached device is a high-speed device. Ports can be disabled by either a fault condition (disconnect event or other fault condition) or by host software. The bit status does not change until the port state actually changes. There might be a delay in disabling or enabling a port due to other host controller and bus events. When the port is disabled downstream propagation of data is blocked on this port, except for reset. This field is zero if Port Power is zero.
30	CSC	Connect Status Change Indicates a change has occurred in the port Current Connect Status. 0 No change 1 Change in Current Connect Status	Default = 0. The Host Controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition, and hub is setting an already-set bit (that is., the bit remains set). Software clears this bit by writing 1 to it. This field is zero if Port Power is zero.
31	CCS	Current Connect Status 0 No device is present 1 Device is present on port	Default = 0. This value reflects the current state of the port, and might not correspond directly to the event that caused EHCI0_PORTSC[CSC] to be set. This field is zero if Port Power is zero.

37.2.6.13 Programmable Microframe Base Value Register (EHCI0_INSNREG0)

This register allows the user to vary the 125µs microframe time for simulation purposes. Each decimal value change in the 1 Microframe Counter Value field corresponds to 1 USB2HClk clock cycle.

Reset: 0x0000_0000

Figure 37-12. Programmable Microframe Base Value Register (EHCI0_INSNREG0)

Bit	Mnemonic	Description	Comments
0:17		Reserved	
18:30	1MFCV	1 Microframe Counter Value	
31	EMFCV	Enable Microframe Counter Value 0 Disabled 1 Use value specified in 1MFCV.	

37.2.6.14 Debug Only Register (EHCI0_INSNREG4)

The ESDPET field scales down the time of specific events within the USB Chirp process for debug purposes.

The EWHCCP and EWHCSP fields allows the user to modify the normally read-only registers EHCI0_HCCPARAMS and EHCI0_HCSPARAMS to accept OPB write transfers.

Reset: 0x0000_0000

Figure 37-13. Debug Only Register (EHCI0_INSNREG4)

Bit	Mnemonic	Description	Comments
0:25		Reserved	
26	NAKRFD	NAK Reload Fix 0 Enable 1 Disable	Incorrect NAK reload transition at the end of a microframe.
27	HCFHD	HC Halted Fix 0 Enable 1 Disable	If ports are suspended and PHY clocks are stopped, the HC Halted Fix bit is set by the Host if the Run/Stop bit is set.
28		Reserved	
29	ESDPET	Enable Scale Down Port Enumeration Time 0 Use defined time 1 Enable scaled down time	
30	EWHCCP	Enable Writable EHCI0_HCCPARAMS register 0 Read Only 1 Writable	
31	EWHCSP	Enable Writable EHCI0_HCSPARAMS register 0 Read Only 1 Writable	

37.2.6.15 ULPI Configuration Register (EHCI0_INSNREG5)

Initiates configuration register operations with the ULPI PHY, enabling the application software to access all ULPI PHY registers as defined in the ULPI Specification.

Reset: 0x0000_0000

Figure 37-14. Debug Only Register (EHCI0_INSNREG5)

Bit	Mnemonic	Description	Comments
0	VC	Vendor Control 0 ULPI transactions complete 1 Load New Command	
1:3		Reserved	
4:7	HPN	Host Port Number	For single port, program this to 0x1.
8:9	OP	Operation 00 Reserved 01 Reserved 10 Register Write 11 Register Read	
10:15	IRA	Immediate Register Address	
16:23	ERA	Extended Register Address	
24:31	RD	Register Data	Write Data for ULPI register. Write and Read Data for ULPI register Read.

37.2.6.16 Revision Register (OHCI0_HCREV)

This register contains the revision number of the OHCI specification supported by this Host Controller.

Reset: 0x0000_0010

Figure 37-15. Revision Register (OHCI0_HCREV)

Bit	Mnemonic	Description	Comments
0:23		Reserved	
24:31	REV	OHCI revision number BCD representation of the version of the HCI specification that is implemented by this Host Controller. For example, a value of 0x11 corresponds to version 1.1.	All of the Host Controller implementations that are compliant with the above specification will have a value of 0x10.

37.2.6.17 Control Register (OHCI0_HCCTRL)

This register defines the operating modes for the Host Controller. With the exception of HCFS and RWC, the fields in this register are modified only by the Host Controller driver.

Reset: 0x0000_0000

Figure 37-16. Control Register (OHCI0_HCCTRL)

Bit	Mnemonic	Description	Comments
0:20		Reserved	
21	RWE	Remote Wake up Enable Used by the Host Controller driver to enable or disable the remote wake up feature upon the detection of upstream resume signaling. 0 Disable remote wake up 1 Enable remote wake up	When this bit is set and OHCI0_HCCMDSTS[RD] is set, a remote wake up is signaled to the Host system. Setting this bit has no impact on the generation of hardware interrupts.
22	RWC	Remote Wake up Connected Indicates whether the Host Controller supports remote wake up signaling. 0 No remote wake up signaling 1 Remote wake up signaling supported	If remote wake up is supported and used by the system it is the responsibility of system firmware to set this bit during POST. The Host Controller clears the bit on a hardware reset but does not alter it on a software reset.
23	IR	Interrupt Routing Determines the routing of interrupts generated by events registered in OHCI0_HCCMDSTS. 0 Interrupts are routed to the normal host bus interrupt mechanism 1 Interrupts are routed to the System Management Interrupt	The Host Controller driver clears this bit on a hardware reset, but it does not alter this bit on a software reset. the Host Controller driver uses this bit as a tag to indicate the ownership of the Host Controller.
24:25	HCFS	Host Controller Functional State 00 USBRESET 01 USBRESUME 10 USBOPERATIONAL 11 USBSUSPEND	A transition to USBOPERATIONAL from another state causes SOF generation to begin 1ms later. The Host Controller driver can determine whether the Host Controller has begun sending SOFs by reading OHCI0_HCCMDSTS[SF]. This field can be changed by the Host Controller only when in the USBSUSPEND state. The Host Controller can move from the USBSUSPEND state to the USBRESUME state after detecting resume signaling from a downstream port. HC enters USBSUSPEND after a software reset, whereas it enters USBRESET after a hardware reset. The latter also resets the root hub and asserts subsequent reset signaling to downstream ports.
26	BLE	Bulk List Enable 0 Disable processing of the Bulk list after the next SOF. 1 Enable processing of the Bulk list in the next Frame	The Host Controller checks this bit whenever it processes the list. When disabled, the Host Controller driver can modify the list. If OHCI0_HCBULKCED is pointing to an endpoint descriptor (ED) to be removed, the Host Controller driver must advance the pointer by updating OHCI0_HCBULKCED before re-enabling processing of the list.

Figure 37-16. Control Register (OHCI0_HCCTRL) (Continued)			
Bit	**Mnemonic**	**Description**	**Comments**
27	CLE	Control List Enable 0 Disable processing of the Control list after the next SOF 1 Enable processing of the Control list in the next Frame	The Host Controller must check this bit whenever it processes the list. When disabled, the Host Controller driver can modify the list. If OHCI0_HCCTRLCED is pointing to an endpoint descriptor to be removed, the Host Controller driver must advance the pointer by updating OHCI0_HCCTRLCED before re-enabling processing of the list.
28	IE	Isochronous Enable Used by the Host Controller driver to enable/ disable processing of isochronous EDs. 0 Disable processing of the periodic list (which now contains only isochronous EDs) and begin processing the Bulk/Control lists 1 Enabled processing the EDs	While processing the periodic list in a Frame, the Host Controller checks the status of this bit when it finds an Isochronous ED (F=1). Setting this bit takes effect in the next Frame, not the current Frame.
29	PLE	Periodic List Enable 0 Disable processing of the Periodic List after the next SOF 1 Enable processing of the Periodic List in the next Frame	The Host Controller must check this bit before it starts processing the list.
30:31	CBSR	Control Bulk Service Ratio Specifies the service ratio (number of Control EDs over Bulk EDs served) between Control and Bulk EDs. 00 1:1 01 2:1 10 3:1 11 4:1	Before processing any of the nonperiodic lists, the Host Controller must compare the ratio specified with its internal count on how many non-empty Control EDs have been processed, in determining whether to continue serving another Control ED or switching to Bulk EDs. The internal count will be retained when crossing the frame boundary. In case of reset, the Host Controller driver is responsible for restoring this value.

37.2.6.18 Command Status Register (OHCI0_HCCMDSTS)

This register is used by the Host Controller to receive commands issued by the Host Controller driver, as well as reflecting the current status of the Host Controller. To the Host Controller driver, it appears to be a "write to set" register. The Host Controller must ensure that bits written as 1 become set in the register while bits written as 0 remain unchanged in the register. The Host Controller driver can issue multiple distinct commands to the Host Controller without concern for corrupting previously issued commands. The Host Controller Driver has normal Read access to all bits.

The SOC field indicates the number of frames with which the Host Controller has detected the scheduling overrun error. This occurs when the Periodic list does not complete before EOF. When a scheduling overrun error is detected, the Host Controller increments the counter and sets OHCI0_HCINTSTS[SO].

Reset: 0x0000_0000

Figure 37-17. Command Status Register (OHCI0_HCCMDSTS)			
Bit	**Mnemonic**	**Description**	**Comments**
0:13		Reserved	

Figure 37-17. Command Status Register (OHCI0_HCCMDSTS) (Continued)

Bit	Mnemonic	Description	Comments
14:15	SOC	Scheduling Overrun Count Incremented on each scheduling overrun error. It is initialized to 0b00 and wraps around at 0b11.	Read only This is incremented when a scheduling overrun is detected even if OHCI0_HCCMDSTS[SO] is already set. It is used by the Host Controller driver to monitor any persistent scheduling problems.
16:27		Reserved	
28	OCR	Ownership Change Request 0 Do not request a change of control of the Host Controller 1 Request a change of control of the Host Controller	R/W When set the Host Controller sets OHCI0_HCCMDSTS[OC]. After the changeover, this bit is cleared and remains so until the next request from Host Controller driver.
29	BLF	Bulk List Filled Used to indicate whether there are any TDs on the Bulk list. 0 Do not process the Bulk list 1 Host Controller driver added a TD to an ED in the Bulk list	R/W When the Host Controller begins processing the head of the Bulk list, it checks BLF. As long as BLF is 0, the Host Controller does not start. If BLF is 1, the Host Controller starts processing the Bulk list and sets BLF to 0. If the Host Controller finds a TD on the list, the Host Controller sets BLF to 1 causing the Bulk list processing to continue. If no TD is found on the Bulk list, and if the Host Controller driver does not set BLF, then BLF is still 0 when the Host Controller completes processing the Bulk list and Bulk list processing stops.
30	CLF	Control List Filled Used to indicate whether there are any TDs on the Control list. 0 Do not process the Control list 1 Host Controller driver added a TD to an ED in	R/W When the Host Controller begins to process the head of the Control list, it checks CLF. As long as CLF is 0, the Host Controller does not start processing the Control list. If CF is 1, the Host Controller starts processing the Control list and sets CLF to 0. If the Host Controller finds a TD on the list, then HC sets CLF to 1 causing the Control list processing to continue. If no TD is found on the Control list, and if the Host Controller driver does not set CLF, then CLF is still 0 when the Host Controller completes processing the Control list and Control list processing stops.
31	HCR	Host Controller Reset 0 Reset of the Host Controller complete 1 Initiate a software reset of the Host Controller	R/W Regardless of the functional state of the Host Controller, it moves to the USBSUSPEND state in which most of the operational registers are reset except those stated otherwise (for example, OHCI0_HCCTRL[IR]), and no Host bus accesses are allowed. The reset operation must be completed within 10ms. This bit, when set, should not cause a reset to the root hub and no subsequent reset signaling should be asserted to its downstream ports.

37.2.6.19 Interrupt Status Register (OHCI0_HCINTSTS)

This register provides status on various events that cause hardware interrupts. When an event occurs, the Host Controller sets the corresponding bit in this register. When a bit becomes set, a hardware interrupt is generated if OHCI0_HCINTE[MIE] is set. The Host Controller driver can clear specific bits in this register by writing 1 to bit positions to be cleared. The Host Controller driver cannot set any of these bits. The Host Controller never clears these bits.

Reset: 0x0000_0000

Figure 37-18. Interrupt Status Register (OHCI0_HCINTSTS)

Bit	Mnemonic	Description	Comments
0		Reserved	
1	OC	Ownership Change Set by the Host Controller when the Host Controller driver sets OHCI0_HCCMDSTS[OCR].	This event, when unmasked, always generates a System Management Interrupt (SMI) immediately. This bit is tied to 0 when the SMI pin is not implemented.
2:24		Reserved	
25	RHSC	Root Hub Status Change Set when the content of OHCI0_HCRHSTS or OHCI0_HCPRSTS has changed.	
26	FNO	Frame Number Overflow Set when the msb of OHCI0_HCFMNUM[FN] changes value, from 0 to 1 or from 1 to 0, and after the Frame Number value has been updated.	
27	UE	Unrecoverable Error 0 Host Controller has been reset 1 Host Controller detected a system error not related to USB	The Host Controller should not do any processing or signaling before the system error has been corrected.
28	RD	Resume Detected 0 USBRESUME state is set 1 Device on the USB is asserting resume signaling	A transition from no resume signaling to resume signaling causes this bit to be set.
29	SF	Start of Frame Set by the Host Controller at each SOF and after the update of the Frame Number.	The Host Controller generates a SOF token at this time.
30	WDH	Write Back Done Head 0 OHCI0_HCDHEAD[DH] has been saved by the Host Controller 1 Done Head value is written to OHCI0_HCDHEAD[DH]	Further updates of the Done Head value do not occur until this bit is cleared.
31	SO	Scheduling Overrun Set when the USB schedule for the current Frame overruns and after the update of the Frame Number.	A scheduling overrun also causes OHCI0_HCCMDSTS[SOC] to be incremented.

37.2.6.20 Interrupt Enable Register (OHCI0_HCINTE)

Each enable bit in this register corresponds to an associated interrupt bit in OHCI0_HCINTSTS. This register is used to control which events generate a hardware interrupt. When a bit is set in OHCI0_HCINTSTS and the corresponding bit in this register is set and MIE is set, a hardware interrupt is requested on the host bus.

Writing 1 to a bit in this register sets the corresponding bit. Writing 0 to a bit in this register leaves the corresponding bit unchanged. On read, the current value of this register is returned.

Offset: 0x0010

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-19. Interrupt Enable Register (OHCI0_HCINTE)

Bit	Mnemonic	Description	Comments
0	MIE	Master Interrupt Enable 0 Ignored 1 Enables interrupt generation due to events specified in the other bits of this register	
1	OC	Ownership Change interrupt enable 0 Ignored 1 Enable interrupt	
2:24		Reserved	
25	RHSC	Root Hub Status Change interrupt enable 0 Ignored 1 Enable interrupt	
26	FNO	Frame Number Overflow interrupt enable 0 Ignored 1 Enable interrupt	
27	UE	Unrecoverable Error interrupt enable 0 Ignored 1 Enable interrupt	
28	RD	Resume Detected interrupt enable 0 Ignored 1 Enable interrupt	
29	SF	Start of Frame interrupt enable 0 Ignored 1 Enable interrupt	
30	WDH	Write Back Done Head interrupt enable 0 Ignored 1 Enable interrupt	
31	SO	Scheduling Overrun interrupt enable 0 Ignored 1 Enable interrupt	

37.2.6.21 Interrupt Disable Register (OHCI0_HCINTDIS)

Each disable bit in this register corresponds to an associated interrupt bit in OHCI0_HCINTSTS. This register is coupled with OHCI0_HCINTE. This register is used to control which events generate a hardware interrupt. When a bit is set in OHCI0_HCINTSTS and the corresponding bit in this register is set and MIE is set, a hardware interrupt is requested on the host bus.

Writing 1 to a bit in this register clears the corresponding bit. Writing 0 to a bit in this register leaves the corresponding bit unchanged. On read, the current value of this register is returned.

Offset: 0x0014

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-20. Interrupt Disable Register (OHCI0_HCINTDIS)

Bit	Mnemonic	Description	Comments
0	MIE	Master Interrupt Disable 0 Ignored 1 Disables interrupt generation due to events specified in the other bits of this register	This field is set after a hardware or software reset.
1	OC	Ownership Change interrupt disable 0 Ignored 1 Disable interrupt	
2:24		Reserved	
25	RHSC	Root Hub Status Change interrupt disable 0 Ignored 1 Disable interrupt	
26	FNO	Frame Number Overflow interrupt disable 0 Ignored 1 Disable interrupt	
27	UE	Unrecoverable Error interrupt disable 0 Ignored 1 Disable interrupt	
28	RD	Resume Detected interrupt disable 0 Ignored 1 Disable interrupt	
29	SF	Start of Frame interrupt disable 0 Ignored 1 Disable interrupt	
30	WDH	Write Back Done Head interrupt disable 0 Ignored 1 Disable interrupt	
31	SO	Scheduling Overrun interrupt disable 0 Ignored 1 Disable interrupt	

37.2.6.22 Host Controller Communications Area Register (OHCI0_HCHCCA)

This register contains the physical address of the Host Controller Communication Area (HCCA). The Host Controller driver determines the alignment restrictions by writing all 1s to this register and then reading the contents. The alignment is evaluated by examining the number of zeroes in the lower order bits. The minimum alignment is 256 bytes. Therefore, bits 0 through 7 must always return 0 when read. This area is used to hold the control structures and the Interrupt table that are accessed by both the Host Controller and the Host Controller driver.

Offset: 0x0018

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-21. Host Controller Communications Area Register (OHCI0_HCHCCA)

Bit	Mnemonic	Description	Comments
0:23	HCCA	Base address of the HCCA.	
24:31		Reserved	

37.2.6.23 Periodic Current Endpoint Descriptor Register (OHCI0_HCPCED)

This register contains the physical address of the current Isochronous or Interrupt Endpoint Descriptor.

Offset: 0x001C

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-22. Periodic Current Endpoint Descriptor Register (OHCI0_HCPCED)

Bit	Mnemonic	Description	Comments
0:27	PCED	Period Current ED Pointer to the head of one of the periodic lists which will be processed in the current Frame.	The content of this register is updated by the Host Controller after a periodic ED has been processed. The Host Controller driver can read the content to determine which ED is being processed at the time of reading.
28:31		Reserved	

37.2.6.24 Control Head Endpoint Descriptor Register (OHCI0_HCCHED)

This register contains the physical address of the first Endpoint Descriptor of the Control list.

Offset: 0x0020

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-23. Control Head Endpoint Descriptor Register (OHCI0_HCCHED)

Bit	Mnemonic	Description	Comments
0:27	CHED	Control Head ED The Host Controller processes the Control list starting with the CHED pointer.	The content is loaded from HCCA during the initialization of the Host Controller .
28:31		Reserved	

37.2.6.25 Control Current Endpoint Descriptor Register (OHCI0_HCCTRLCED)

This register contains the physical address of the current Endpoint Descriptor of the Control list.

Offset: 0x0024

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-24. Control Current Endpoint Descriptor Register (OHCI0_HCCTRLCED)

Bit	Mnemonic	Description	Comments
0:27	CCED	Control Current ED Initially, this is set to zero to indicate the end of the Control list and no processing occurs. Otherwise, the Host Controller reads the instantaneous value of this register. This pointer is advanced to the next ED after serving the present one. The Host Controller continues processing the list from where it left off in the last Frame. When it reaches the end of the Control list, it checks OHCI0_HCCMDSTS[CLF]. If set, it copies the content of OHCI0_HCCHED to OHCI0_HCCTRLCED and clears OHCI0_HCCMDSTS[CLF].	The Host Controller driver can modify this register only when OHCI0_HCCTRL[CLE] = 0.
28:31		Reserved	

37.2.6.26 Bulk Head Endpoint Descriptor Register (OHCI0_HCBULKHED)

This register contains the physical address of the first Endpoint Descriptor of the Bulk list.

Offset: 0x0028

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-25. Bulk Head Endpoint Descriptor Register (OHCI0_HCBULKHED)

Bit	Mnemonic	Description	Comments
0:27	BHED	Bulk Head ED The Host Controller processes the Bulk list starting with the BHED pointer.	The content is loaded from HCCA during the initialization of the Host Controller.
28:31		Reserved	

37.2.6.27 Bulk Current Endpoint Descriptor Register (OHCI0_HCBULKCED)

This register contains the physical address of the current endpoint of the Bulk list. As the Bulk list is processed in a round-robin fashion, the endpoints are ordered according to their insertion to the list.

Offset: 0x002C

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-26. Bulk Current Endpoint Descriptor Register (OHCI0_HCBULKCED)

Bit	Mnemonic	Description	Comments
0:27	BCED	Bulk Current ED Initially, this is set to zero to indicate the end of the Bulk list and no processing occurs. Otherwise, the Host Controller reads the instantaneous value of this register. This pointer is advanced to the next ED after serving the present one. The Host Controller continues processing the list from where it left off in the last Frame. When it reaches the end of the Bulk list, it checks OHCI0_HCCMDSTS[BLF]. If set, it copies the content of OHCI0_HCBULKHED to OHCI0_HCBULKCED and clears OHCI0_HCCMDSTS[BLF].	The Host Controller driver can modify this register only when OHCI0_HCCTRL[BLE] = 0.
28:31		Reserved	

37.2.6.28 Done Head Transfer Descriptor Register (OHCI0_HCDHEAD)

This register contains the physical address of the last completed Transfer Descriptor (TD) that was added to the Done queue. In normal operation, the Host Controller driver should not need to read this register as its content is periodically written to the HCCA.

Offset: 0x0030

Size (bits): 32

Access: Read only

Reset: 0x0000_0000

Figure 37-27. Done Head Transfer Descriptor Register (OHCI0_HCDHEAD)

Bit	Mnemonic	Description	Comments
0:27	DH	Done Head When a TD is completed, the Host Controller writes the content of this register to the NextTD field of the TD. The Host Controller then overwrites the content of this register with the address of this TD.	This is set to 0 whenever the Host Controller writes the content of this register to HCCA. It also sets OHCI0_HCINTSTS[WDH].
28:31		Reserved	

37.2.6.29 Frame Interval Register (OHCI0_HCFMINT)

This register contains a 14-bit value which indicates the bit time interval in a Frame, (that is, the data between two consecutive SOFs), and a 15-bit value indicating the Full Speed maximum packet size that the Host Controller can transmit or receive without causing scheduling overrun.

The Host Controller Driver can carry out minor adjustments on the Frame Interval by writing a new value over the present one at each SOF. This provides the programmability necessary for the Host Controller to synchronize with an external clocking resource and to adjust for any unknown local clock offset.

Offset: 0x0034

Size (bits): 32

Access: R/W

Reset: 0x0000_2EDF

Figure 37-28. Frame Interval Register (OHCI0_HCFMINT)

Bit	Mnemonic	Description	Comments
0	FIT	Frame Interval Toggle Toggles whenever a new value is loaded in Frame Interval.	
1:15	FSMPS	FS Largest Data Packet Specifies the value loaded into the Largest Data Packet Counter at the beginning of each frame.	The counter value represents the largest number of data in bits which can be sent or received by the Host Controller in a single transaction at any given time without causing scheduling overrun. The field value is calculated by the Host Controller.
16:17		Reserved	
18:31	FI	Frame Interval Specifies the interval between two consecutive SOFs in bit times. The nominal value is set to 11,999.	The Host Controller driver should store the current value of this field before resetting the Host Controller. Setting OHCI0_HCCMDSTS[HCR] causes the Host Controller to reset this field to its nominal value. The Host Controller driver can choose to restore the stored value upon the completion of the reset sequence.

37.2.6.30 Frame Remaining Register (OHCI0_HCFMREM)

This register is a 14-bit down counter showing the bit time remaining in the current Frame.

Offset: 0x0038

Size (bits): 32

Access: Read only

Reset: 0x0000_0000

Figure 37-29. Frame Remaining Register (OHCI0_HCFMREM)

Bit	Mnemonic	Description	Comments
0	FRT	Frame Remaining Toggle Set whenever FR reaches 0.	Used by the Host Controller driver for synchronization between OHCI0_HCFMINT[FI] and OHCI0_HCFMREM[FR].
1:17		Reserved	
18:31	FR	Frame Remaining Counter is decremented at each bit time. When it reaches 0, it is reset by loading the value specified in OHCI0_HCFMINT[FI] at the next bit time boundary.	When entering the USBOPERATIONAL state, The Host Controller driver loads FR with OHCI0_HCFMINT[FI] and uses the updated value from the next SOF.

37.2.6.31 Frame Number Register (OHCI0_HCFMNUM)

This register is a 16-bit counter. It provides a timing reference among events happening in the Host Controller and the Host Controller driver. The Host Controller driver can use the 16-bit value specified in this register and generate a 32-bit frame number without requiring frequent access to the register.

Offset: 0x003C

Size (bits): 32

Access: Read only

Reset: 0x0000_0000

Figure 37-30. Frame Number Register (OHCI0_HCFMNUM)

Bit	Mnemonic	Description	Comments
0:15		Reserved	
16:31	FN	Frame Number Incremented when OHCI0_HCFMREM[FR] is loaded.	When entering the USBOPERATIONAL state, this field is incremented automatically. The content is written to HCCA after the Host Controller has incremented FN at each frame boundary and sent a SOF but before the Host Controller reads the first ED in that Frame. After writing to HCCA, the Host Controller sets OHCI0_HCINTSTS[SF].

37.2.6.32 Periodic Start Register (OHCI0_HCPRDSTRT)

This register has a 14-bit programmable value which determines the earliest time the Host Controller should start processing the periodic list.

Offset: 0x0040

Size (bits): 32

Access: R/W

Reset: 0x0000_0000

Figure 37-31. Periodic Start Register (OHCI0_HCPRDSTRT)

Bit	Mnemonic	Description	Comments
0:17		Reserved	
18:31	PS	Periodic Start Cleared by a hardware reset. Set by Host Controller initialization	The value is calculated from OHCI0_HCFMINT[FI]. It can be in error by 10%. A typical value is 0x3E67. When OHCI0_HCFMREM[FR] reaches the value specified, processing of the periodic lists has priority over Control/Bulk processing. The Host Controller starts processing the Interrupt list after completing the current Control or Bulk transaction that is in progress.

37.2.6.33 Low Speed Threshold Register (OHCI0_HCLSTHRES)

This register contains an 11-bit value used by the Host Controller to determine whether to commit to the transfer of a maximum 8-byte LS packet before EOF. Neither the Host Controller nor the Host Controller driver are allowed to change this value.

Offset: 0x0044

Size (bits): 32

Access: R/W

Reset: 0x0000_0628

Figure 37-32. Low Speed Threshold Register (OHCI0_HCLSTHRES)

Bit	Mnemonic	Description	Comments
0:19		Reserved	
20:31	LST	Low Speed Threshold Contains a value which is compared to OHCI0_HCFMREM[FR] prior to initiating a Low Speed transaction. The transaction is started only if OHCI0_HCFMREM[FR] ≥ LST.	This value is calculated by the Host Controller driver and includes consideration of transmission and setup overhead.

37.2.6.34 Root Hub Descriptor A Register (OHCI0_HCRHDESCA)

This register is the first of two (A and B) describing the characteristics of the Root Hub. Reset values are implementation-specific. The descriptor length, descriptor type, and hub controller current fields of the hub Class Descriptor are emulated by the Host Controller driver. All other fields are located in the OHCI0_HCRHDESCA and OHCI0_HCRHDESCB registers.

Offset: 0x0048

Size (bits): 32

Access: See field specification

Reset: 0x0200_0001

Figure 37-33. Root Hub Descriptor A Register (OHCI0_HCRHDESCA)

Bit	Mnemonic	Description	Comments
0:7	POTPGT	Power-on to Power-Good Time Specifies the duration the Host Controller driver has to wait before accessing a powered-on port of the Root Hub.	R/W The unit of time is 2ms so the duration is calculated as POTPGT x 2ms.
8:18		Reserved	R/W
19	NOCP	No Over Current Protection Describes how over current status for the Root Hub ports is reported. 0 Over Current status is collectively reported for all downstream ports 1 No Over Current protection supported	R/W When NOCP = 0, OCPM specifies global or per-port reporting.
20	OCPM	Over Current Protection Mode Describes how the over current status for the Root Hub ports are reported. 0 Over Current status is reported collectively for all downstream ports 1 Over Current status is reported on a per-port basis	R/W OCPM is valid only if NOCP = 0. After reset, OCPM should reflect the same mode as PSM.
21	DT	Device Type Specifies that the Root Hub is not a compound device.	Read on The Root Hub is not permitted to be a compound device so DT should always = 0
22	NPS	No Power Switching Specifies whether power switching is supported or ports are always powered. 0 Ports are power switched 1 Ports are always powered on when the Host Controller is powered on	R/W When NPS = 0, PSM specifies global or per-port switching.

Figure 37-33. Root Hub Descriptor A Register (OHCI0_HCRHDESCA) (Continued)			
Bit	**Mnemonic**	**Description**	**Comments**
23	PSM	Power Switching Mode Specifies the power switching of the Root Hub ports is controlled. 0 All ports are powered at the same time 1 Each port is powered individually. This mode allows port power to be controlled by either the global switch or the per-port switching. If OHCI0_HCRHDESCB[PPCM] is set, the port responds only to port power commands (Set/ClearPortPower). If the port mask is cleared, then the port is controlled only by the global power switch (Set/ClearGlobalPower).	R/W This field is valid only if NPS = 0.
24:31	NDP	Number of Downstream Ports These bits specify the number of downstream ports supported by the Root Hub. Minimum = 1 Maximum = 15.	Read only

37.2.6.35 Root Hub Descriptor B Register (OHCI0_HCRHDESCB)

This register is the second of two (A and B) describing the characteristics of the Root Hub. These fields are written during initialization to correspond with the system implementation. Reset values are implementation-specific.

Reset: 0x0000_0000

Figure 37-34. Root Hub Descriptor B Register (OHCI0_HCRHDESCB)			
Bit	**Mnemonic**	**Description**	**Comments**
0:15	PPCM	Port Power Control Mask Each bit indicates if a port is affected by a global power control command when PSM is set. When 0x0, port is controlled by the global power switch (Set/ClearGlobalPower). Otherwise port's power state is affected only by per-port power control (Set/ClearPortPower). Bit 0: Reserved Bit 1: Ganged-power mask on Port #1 Bit 2: Ganged-power mask on Port #2 ... Bit 15: Ganged-power mask on Port #15	If the device is configured for global switching mode (PSM = 0), this field is not valid.
16:31	DR	Device Removable Each bit is dedicated to a port of the Root Hub. When 0x0, the attached device is removable. Otherwise, the attached device is not removable. Bit 0: Reserved Bit 1: Device attached to Port #1 Bit 2: Device attached to Port #2 ... Bit 15: Device attached to Port #15	

37.2.6.36 Root Hub Status Register (OHCI0_HCRHSTS)

This register is divided into two parts. The lower word represents the Hub Status field and the upper word represents the Hub Status Change field.

Access: See field specification

Reset: 0x0000_0000

Figure 37-35. Root Hub Status Register (OHCI0_HCRHSTS)

Bit	Mnemonic	Description	Comments
0	CRWE	Clear Remote Wake up Enable Writing 1 clears DRWE. Writing 0 has no effect.	Read with Clear
1:13		Reserved	
14	OCIC	Over Current Indicator Change Set by hardware when a change has occurred to OCI. The Host Controller driver clears this bit by writing 1. Writing 0 has no effect.	R/W
15	LPSC	Local Power Status Change Set to turn power on to all ports (clear OHCI0_HCPRSTS[PPS]) when in global power mode (PSM = 0). In per-port power mode, it sets OHCI0_HCPRSTS[PPS] only on ports whose OHCI0_HCRHDESCB[PPCM] bit is not set. Writing 0 has no effect.	R/W The Root Hub does not support the local power status feature and this bit is always read as 0.
16	DRWE	Device Remote Wake up Enable Enables a OHCI0_HCPRSTS[CSC] as a resume event, causing a USBSUSPEND to USBRESUME state transition and setting the ResumeDetected interrupt. 0 OHCI0_HCPRSTS[CSC] is not a remote wake up event 1 OHCI0_HCPRSTS[CSC] is a remote wake up event Writing 1 sets DRWE. Writing 0 has no effect.	R/W
17:29		Reserved	
30	OCI	Over Current Indicator This bit is set by hardware when a change has occurred to the OCI field of this register. The Host Controller driver clears this bit by writing 1. Writing 0 has no effect.	R/W
31	LPS	Local Power Status Set to turn off power to all ports (clear OHCI0_HCPRSTS[PPS]) when in global power mode (PSM = 0). In per-port power mode, it clears OHCI0_HCPRSTS[PPS] only on ports whose OHCI0_HCRHDESCB[PPCM] bit is not set. Writing 0 has no effect.	R/W The Root Hub does not support the local power status feature and this bit is always read as 0.

37.2.6.37 Root Hub Port Status Register (OHCI0_HCPRSTS)

This register is used to control and report port events on a per-port basis. *OHCI0_HCRHDESCA*[NDP] represents the number of these registers that are implemented in hardware. The lower word is used to reflect the port status; the upper word reflects the status change bits. If a transaction (token through handshake) is in progress when a write to change port status occurs, the resulting port status change must be postponed until the transaction completes.

Reset: 0x0000_0000

Figure 37-36. Root Hub Port Status Register (OHCI0_HCPRSTS)

Bit	Mnemonic	Description	Comments
0:10		Reserved	
11	PRSC	Port Reset Status Change Indicates end of the 10ms port reset signal. 0 Port reset not complete 1 Port reset complete	The Host Controller driver writes 1 to clear this bit. Writing 0 has no effect.
12	OCIC	Port Over Current Indicator Change Root Hub changed POCI. 0 No change in POCI 1 POCI has changed	This bit is valid only if over current conditions are reported on a per-port basis. The Host Controller driver writes 1 to clear this bit. Writing 0 has no effect.
13	PSSC	Port Suspend Status Change Indicates a full resume sequence has been completed. 0 Resume not completed 1 Resume completed	This sequence includes the 20s resume pulse, and the 3ms resychronization delay. The Host Controller driver writes 1 to clear this bit. Writing 0 has no effect. This bit is also cleared when PRSC is set.
14	PESC	Port Enable Status Change Indicates a hardware event cause PES to be cleared. 0 No change in PES 1 Change in PES	Changes from the Host Controller driver do not set this bit. The Host Controller driver writes a 1 to clear this bit. Writing a 0 has no effect.
15	CSC	Connect Status Change Indicates a connect or disconnect event occurred. 0 No change in CCS 1 Change in CCS	The Host Controller driver writes 1 to clear this bit. Writing 0 has no effect. If CCS is cleared when a SetPortReset, SetPortEnable, or SetPortSuspend write occurs, this bit is set to force the driver to re-evaluate the connection status since these writes should not occur if the port is disconnected. **Note:** If OHCI0_HCRHDESCB[DR] is set, this bit is set only after a Root Hub reset to inform the system that the device is attached.
16:21		Reserved	
22	LSDA	Low Speed Device Attached Indicates the speed of the device attached to this port 0 Full-speed device attached 1 Low-speed device attached	This field is valid only when CCS is set. The Host Controller driver clears the PPS by writing 1 to this bit. Writing 0 has no effect.

Figure 37-36. Root Hub Port Status Register (OHCI0_HCPRSTS) (Continued)

Bit	Mnemonic	Description	Comments
23	PPS	Port Power Status Reflects the port's power status 0 Port power is off 1 Port power is on	This bit is cleared if an over current condition is detected. Which power control switches are enabled is determined by OHCI0_HCRHDESCA[PSM] and OHCI0_HCRHDESCB[PPCM]. In global switching mode (PSM = 0), only Set/ClearGlobalPower controls this bit. In per-port power switching (PSM = 1), if OHCI0_HCRHDESCB[PPCM] for the port is set, only Set/ClearPortPower commands are enabled. If the mask is not set, only Set/ClearGlobalPower commands are enabled. When port power is disabled, CCS, PES, PSS, and PRS should be reset. The Host Controller driver writes 1 to set the this bit. Writing 0 has no effect.
24:26		Reserved	
27	PRS	Port Reset Status Indicates status of port reset signal. 0 Port reset signal is not active 1 Port reset signal is active	When this bit is set, port reset signaling is asserted. When reset is completed, this bit is cleared when PRSC is set. This bit cannot be set if CCS is cleared. The Host Controller driver sets the port reset signaling by writing 1 to this bit. Writing 0 has no effect. If CCS is cleared, this write does not set this bit, but instead sets CSC. This informs the driver that it attempted to reset a disconnected port.
28	POCI	Port Over Current Indicator Reflects the over current input signal 0 No over current condition 1 Over current condition detected	This bit is only valid when the Root Hub is configured in such a way that over current conditions are reported on a per-port basis. If per-port over current reporting is not supported, this bit is set to 0. The Host Controller driver writes 1 to initiate a resume. Writing 0 has no effect. A resume is initiated only if PSS is set.
29	PSS	Port Suspend Status Indicates the port is suspended or in the resume sequence. 0 Port is not suspended 1 Port is suspended	PSS is set by a port suspend and cleared when PSSC is set at the end of the resume interval. This bit cannot be set if CCS is cleared. This bit is also cleared when PRSC is set at the end of the port reset or when the Host Controller is placed in the USBRESUME state. If an upstream resume is in progress, it should propagate to the Host Controller. The Host Controller driver sets PSS bit by writing 1. Writing 0 has no effect. If CCS is cleared, this write does not set PPSS; instead it sets CSC. This informs the driver that it attempted to suspend a disconnected port.
30	PES	Port Enable Status Indicates whether the port is enabled or disabled. 0 Port is disabled 1 Port is enabled	The Root Hub can clear this bit when an over current condition, disconnect event, switched-off power, or operational bus error such as babble is detected. This change also causes PESC to be set. PES cannot be set when CCS is cleared. PES is also set at the completion of a port reset when PRSC is set, or a port suspend when PSSC is set. The Host Controller driver sets PES by writing 1. Writing 0 has no effect. If CCS is cleared, this write does not set PES, but instead sets CCS. This informs the driver that it attempted to enable a disconnected port.

Figure 37-36. Root Hub Port Status Register (OHCI0_HCPRSTS) (Continued)

Bit	Mnemonic	Description	Comments
31	CCS	Current Connect Status Reflects the current state of the downstream port. 0 No device connected 1 Device connected The Host Controller driver writes 1 to this bit to clear PES. Writing 0 has no effect.	This bit is always read as 1 when the attached device is non-removable

38. USB 2.0 On-The-Go (OTG) Controller (PPC460EX/EXr only)

The following sections provide information on the USB 2.0 OTG interface. USB is available in the PPC460EX/EXr only.

38.1 USB Overview

The USB 2.0 OTG interface supports both Device and Host functions and is fully compliant with the *On-The-Go Supplement to the USB 2.0 Specification*, Revision 1.0a. It can also be configured as a Host-only or Device-only controller, fully compliant with the *USB 2.0 Specification.* The USB 2.0 OTG configurations support high speed (HS, 480 Mbps), full speed (FS, 12 Mbps), and low speed (LS, 1.5 Mbps Host only) transfers. Additionally, USB 2.0 OTG can be configured for USB 1.1 FS and LS. USB 2.0 OTG connects to the PLB4 crossbar switch through the industry-standard AMBA High- Performance Bus (AHB) bridge internally for communicating with the application and system memory, and is fully compliant with the *AMBA Specification*, Revision 2.0. USB 2.0 OTG is optimized for the following applications and systems:

- Portable electronic devices
- Point-to-point applications (no hub, direct connection to HS, FS, or LS Device)
- Multi-point applications (as an embedded USB Host) to devices (hub and split support)

The USB 2.0 OTG Controller includes the following features by category:

- General features
  - Different clocks for AHB and the PHY interfaces
  - Slave or internal DMA modes to optimize performance and reduce the CPU load
  - USB power management features
  - Packet based, dynamic FIFO memory allocation for Endpoints for small FIFOs and flexible, efficient use of RAM
  - Maximum packet size of 1024B (isochronous) and 512B (bulk)
  - Change an Endpoint's FIFO memory size during transfers
  - Endpoint FIFO sizes that are not powers of two, to allow the use of contiguous memory locations
  - Isochronous traffic: three packets per microframe (196.6Mbps throughput)
  - Hardware registers in the Host and Device modes
  - Keep Alive in LS mode and SOFs in HS and FS modes
- Software features
  - Hardware assist for Device mode non-periodic IN sequencing
  - Handles USB commands (SETUP transactions are detected and their command payloads are forwarded to the application for decoding).
  - Handles USB errors.
- Application features
  - Interfaces for the application by means of the AHB:
    - AHB Slave interface for accessing control and status registers (CSRs), the data FIFO, and queues
    - AHB Master interface for data FIFO access when internal DMA is enabled
  - Only 32-bit data on the AHB
  - All AHB burst types in AHB Slave interface

- USB 2.0 supported features
  - ULPI interface—8-bit SDR, external PHY
  - Session Request Protocol (SRP)
  - Host Negotiation Protocol (HNP)
  - Up to two bidirectional Endpoints, including control Endpoint 0
  - Up to four Host channels.
  - Generic root hub
  - No multipoint support
  - Automatic ping capabilities
- Power Optimization features
  - PHY clock gating during USB Suspend mode and Session-Off mode
  - AHB clock gating during USB Suspend mode and Session-Off mode
  - Partial power-off during USB Suspend mode and Session-Off mode

38.2 System Description

The USB OTG provides two architectural modes of operation: Slave-only mode and Internal DMA mode.

Slave mode is typically selected when there is enough CPU bandwidth to execute the USB driver and to move data between memory and the USB function. Data transfer between the Host and the system memory is handled by the application.

Internal DMA mode is typically selected when the CPU bandwidth to process the USB transfer is limited. In this mode, an internal DMA controller transfers data between the memory and USB. The driver sets up the transfer and the USB OTG interrupts the processor only on transfer completion or an error condition.

38.2.1 Shared FIFO Operation

The USB shares a single SPRAM between transmit (periodic and non-periodic) and receive FIFOs. The size of each transmit and receive FIFO can be programmed dynamically.

The Host uses one transmit FIFO for all non-periodic OUT transactions and one transmit FIFO for all periodic OUT transactions. These transmit FIFOs are used as transmit buffers to hold the data (payload of the transmit packet) to be transmitted. The Host pipes the USB transactions through Request queues (one for periodic and one for non-periodic). Each entry in the Request queue holds the IN or OUT channel number along with other information to perform a transaction. The order in which the requests are written into the queue determines the sequence of transactions. The Host processes the periodic Request queue first, followed by the non-periodic Request queue, at the beginning of each (micro)frame. The Host uses one receive FIFO for all periodic and non-periodic transactions. The FIFO is used as a receive buffer to hold the received data (payload of the received packet) until it is transferred to the system memory. The status of each packet received also goes into the FIFO. The status entry holds the IN channel number along with other information, such as received byte count and validity status, to perform a transaction on the AHB.

In Device mode, the function is configured to have a shared transmit FIFO for all non-periodic IN endpoints. In the shared FIFO architecture, the application must predict the order in which the Host sends IN tokens and program the Endpoints accordingly. Additionally, when packet transmission results in an error, the function generates an interrupt and the application must re-enable the Endpoint. The USB Device uses a single transmit FIFO to store the

data for all non-periodic Endpoints and one transmit FIFO per periodic Endpoint to store the data to be transmitted in the next microframe. The data is fetched by DMA or written by the application into the transmit FIFOs and is transmitted when the IN token is received.

The Request queue contains the number of the Endpoint for which the data is written into the data FIFO. The application must predict the order in which the Host accesses the non-periodic Endpoints and must write the data into the non-periodic FIFO accordingly. Because each Periodic IN Endpoints has its own FIFO, no order prediction is needed for periodic IN transfers. The USB Device uses a single receive FIFO to receive the data for all the OUT Endpoints. The receive FIFO holds the status of the received data packet, such as byte count, data PID and the validity of the received data. The DMA or application reads the data out of the receive FIFO as it is received.

38.2.2 ULPI Interface Operation

The ULPI interface connects to an external high-speed PHY.

Table 38-1. ULPI Interface Signals

Signal Name	I/O	Description
USB2DClk	I	ULPI clock. Receives the 60MHz clock supplied by the high-speed ULPI PHY.
USB2DDir	I	Data bus direction control. When the PHY has data to transfer to the USB OTG Controller, it drives USB2DDir high to take ownership of the bus. When the PHY has no data to transfer, it drives USB2DDir low and monitors the bus. 0 USB OTG is the driver 1 ULPI PHY is the driver
USB2DNext	I	Next data control. When the USB OTG Controller is sending data to the PHY, USB2DNext indicates when the current byte is accepted by the PHY. The USB OTG Controller places the next byte on the data bus in the following clock cycle. When the PHY is sending data to the USB OTG Controller, USB2DNext indicates when a new byte is available for the USB OTG Controller to consume.
USB2DD0:7	I/O	ULPI data I/O. ULPI data input to the function driven by the PHY. ULPI data output bus driven by the USB OTG Controller.
USB2DStop	O	Stop output control. The USB OTG Controller asserts this signal for one clock cycle to indicate the end of a USB transmit packet or a register write operation and, optionally, to stop packet reception. Assertion occurs after the last data byte is presented on the bus.

38.3 Control and Status Overview

By reading from and writing to the Control and Status Registers (CSRs) through the AHB Slave interface, your application controls the USB function. These registers are 32 bits wide, and the addresses are 32-bit block aligned. CSRs are classified as follows:.

- Global Registers
- Host Mode Registers
  - Host Global Registers
  - Host Port CSRs
  - Host Channel-Specific Registers

- Device Mode Registers
  - Device Global Registers
  - Device Endpoint-Specific Registers
- Power and Clock-Gating Registers
- Data FIFO (DFIFO) Access Registers

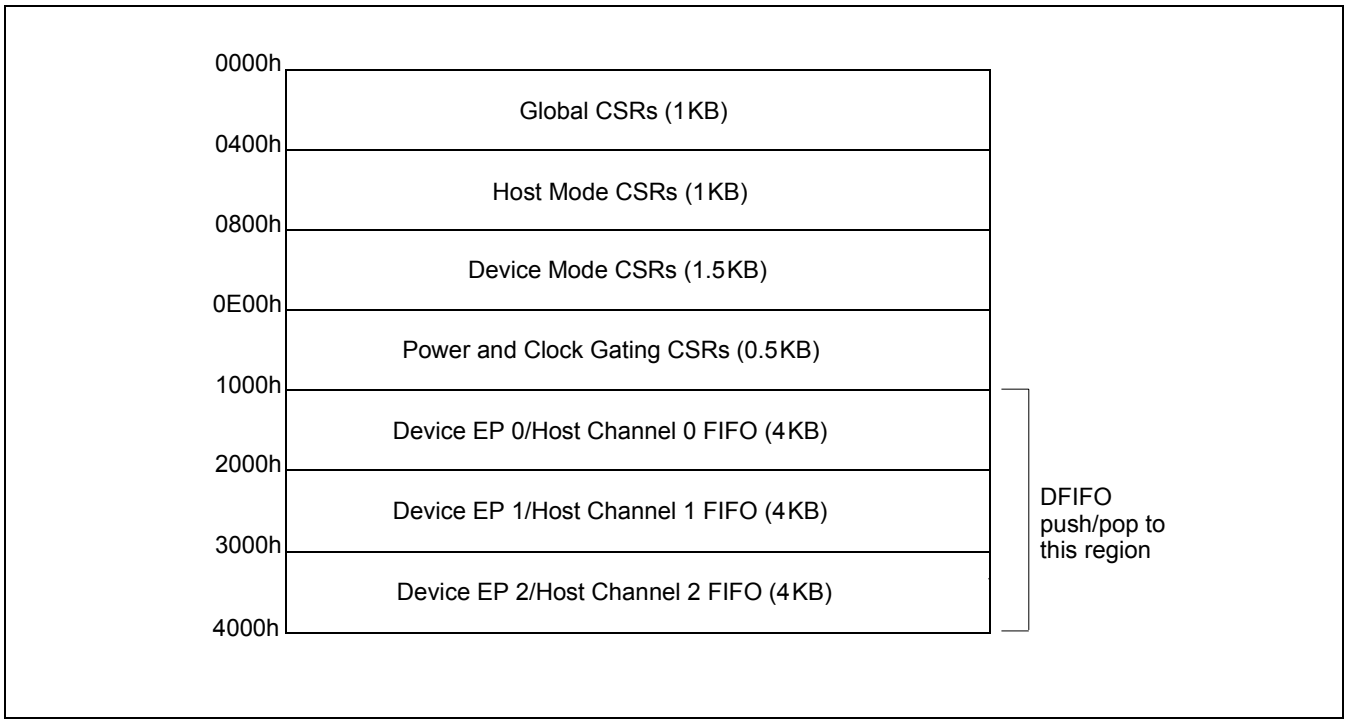
Only the Global, Clock Gating, Data FIFO Access, and Host Port registers can be accessed in both Host and Device modes.

When the function is operating in one mode, either Device or Host, the application must not access registers from the other mode. If an illegal access occurs, a Mode Mismatch interrupt is generated and reflected in the Interrupt register. When the function switches from one mode to another, the registers in the new mode of operation must be reprogrammed as they would be after a power-on reset.

38.4 CSR Memory Map

The following figure shows the CSR address map. The CSR base address is 0x4BFF80000.

Figure 38-1. OTG CSR Memory Map



38.4.1 Global CSRs

These registers are available in both Host and Device modes. The following table gives the offset from the base address 0x4BFF80000.

Table 38-2. Global CSRs

Mnemonic	Register	Offset	Page
USB0_GOTGCTL	OTG Control and Status Register	0x000	1341
USB0_GOTGINT	OTG Interrupt Register	0x004	1342
USB0_GAHBCFG	AHB Configuration Register	0x008	1345
USB0_GUSBCFG	USB Configuration Register	0x00C	1345
USB0_GRSTCTL	Reset Register	0x010	1346
USB0_GINTSTS	Interrupt Register	0x014	1349
USB0_GINTMSK	Interrupt Mask Register	0x018	1354
USB0_GRXSTSR	Receive Status Debug Read Register (Read Only)	0x01C	1355
USB0_GRXSTSP	Receive Status Read/Pop Register (Read Only)	0x020	1355
USB0_GRXFSIZ	Receive FIFO Size Register	0x024	1356
USB0_GNPTXFSIZ	Non-periodic Transmit FIFO Size Register	0x028	1357
USB0_GNPTXSTS	Non-periodic Transmit FIFO/Queue Status Register (Read Only)	0x02C	1357
USB0_GPVNDCTL	PHY Vendor Control Register	0x034	1358
USB0_GSNPSID	ID Register (Read Only)	0x040	1359
USB0_HPTXFSIZ	Host Periodic Transmit FIFO Size Register	0x100	1360
USB0_DPTXFSIZn	Device Periodic Transmit FIFO n Size Register	0x104–0x13C	1360

38.4.2 Host Mode CSRs

These registers must be programmed every time the function changes to Host mode. The following table gives the offset from the base address 0x4BFF80000.

Table 38-3. Host Mode CSRs

Mnemonic	Register	Offset	Page
USB0_HCFG	Host Configuration Register	0x400	1361
USB0_HFIR	Host Frame Interval Register	0x404	1362
USB0_HFNUM	Host Frame Number/Frame Time Remaining Register	0x408	1362
USB0_HPTXSTS	Host Periodic Transmit FIFO/Queue Status Register	0x410	1363

Table 38-3. Host Mode CSRs (Continued)

Mnemonic	Register	Offset	Page
USB0_HAINT	Host All Channels Interrupt Register	0x414	1363
USB0_HAINTMSK	Host All Channels Interrupt Mask Register	0x418	1364
USB0_HPRT	Host Port Control and Status Register	0x440	1364
USB0_HCCHARn	Host Channel n Characteristics Register	0x500	1366
USB0_HCSPLTn	Host Channel n Split Control Register	0x504	1367
USB0_HCINTn	Host Channel n Interrupt Register	0x508	1368
USB0_HCINTMSKn	Host Channel n Interrupt Mask Register	0x50C	1369
USB0_HCTSIZn	Host Channel n Transfer Size Register	0x510	1370
USB0_HCDMAn	Host Channel n DMA Address Register	0x514	1370
	Host Channel 1 Registers	0x520–0x53C	na
	Host Channel 2 Registers	0x540–0x55C	na

38.4.3 Device Mode CSRs

These registers must be programmed every time the function changes to Device mode. The following table gives the offset from the base address 0x4BFF80000.

Table 38-4. Device Mode CSRs

Mnemonic	Register	Offset	Page
USB0_DCFG	Device Configuration Register	0x800	1371
USB0_DCTL	Device Control Register	0x804	1372
USB0_DSTS	Device Status Register (Read Only)	0x808	1374
USB0_DIEPMSK	Device IN Endpoint Common Interrupt Mask Register	0x810	1375
USB0_DOEPMSK	Device OUT Endpoint Common Interrupt Mask Register	0x814	1375
USB0_DAINT	Device All Endpoints Interrupt Register	0x818	1376
USB0_DAINTMSK	Device All Endpoints Interrupt Mask Register	0x81C	1376
USB0_DTKNQR1	Device IN Token Sequence Learning Queue Read Register 1 (Read Only)	0x820	1377
USB0_DTKNQR2	Device IN Token Sequence Learning Queue Read Register 2 (Read Only)	0x824	1378
USB0_DTKNQR3	Device IN Token Sequence Learning Queue Read Register 3 (Read Only)	0x830	1378
USB0_DTKNQR4	Device IN Token Sequence Learning Queue Read Register 4 (Read Only)	0x834	1378
USB0_DVBUSDIS	Device VBUS Discharge Time Register	0x828	1378
USB0_DVBUSPULSE	Device VBUS Pulsing Time Register	0x82C	1379

Table 38-4. Device Mode CSRs (Continued)

Mnemonic	Register	Offset	Page
USB0_DTHRCTL	Device Threshold Control Register	0x830	
USB0_DIEPEMPMSK	Device IN Endpoint FIFO Empty Interrupt Mask Register	0x834	1380
USB0_DIEPCTL0	Device Control IN Endpoint 0 Control Register	0x900	1380
USB0_DIEPCTLn	Device Control IN Endpoint n Control Register	0x900	1383
USB0_DIEPINTn	Device IN Endpoint n Interrupt Register	0x908	1386
USB0_DIEPTSIZ0	Device IN Endpoint 0 Transfer Size Register	0x910	1387
USB0_DIEPTSIZn	Device IN Endpoint n Transfer Size Register	0x920	1388
USB0_DIEPDMAn	Device IN Endpoint n DMA Address Register	0x914	1389
USB0_DTXFSTSn	Device IN Endpoint Transmit FIFO Status Register	0x918	1390
	Device IN Endpoint 1 Registers	0x920–0x93C	na
	Device IN Endpoint 2 Registers	0x940–0x95C	na
USB0_DOEPCTL0	Device Control OUT Endpoint 0 Control Register	0xB00	1382
USB0_DOEPINT0	Device OUT Endpoint 0 Interrupt Register	0xB08	1386
USB0_DOEPTSIZ0n	Device OUT Endpoint 0 Transfer Size Register	0xB10	1387
USB0_DOEPTSIZn	Device OUT Endpoint n Transfer Size Register	0xB30	1388
USB0_DOEPDMAn	Device OUT Endpoint 0 DMA Address Register	0xB14	1389
	Device OUT Endpoint 1 Registers	0xB20–0xB3C	na
	Device OUT Endpoint 2 Registers	0xB40–0xB5C	na

38.4.4 Clock Gating

There is a single register for clock gating. It is available in both Host and Device modes. The following table gives the offset from the base address 0x4BFF80000.

Table 38-5. Clock Gating Register

Mnemonic	Register	Offset	Page
USB0_PCGCTL	Clock Gating Control Register	0xE00	1390

38.4.5 Data FIFO (DFIFO) Access Register Map

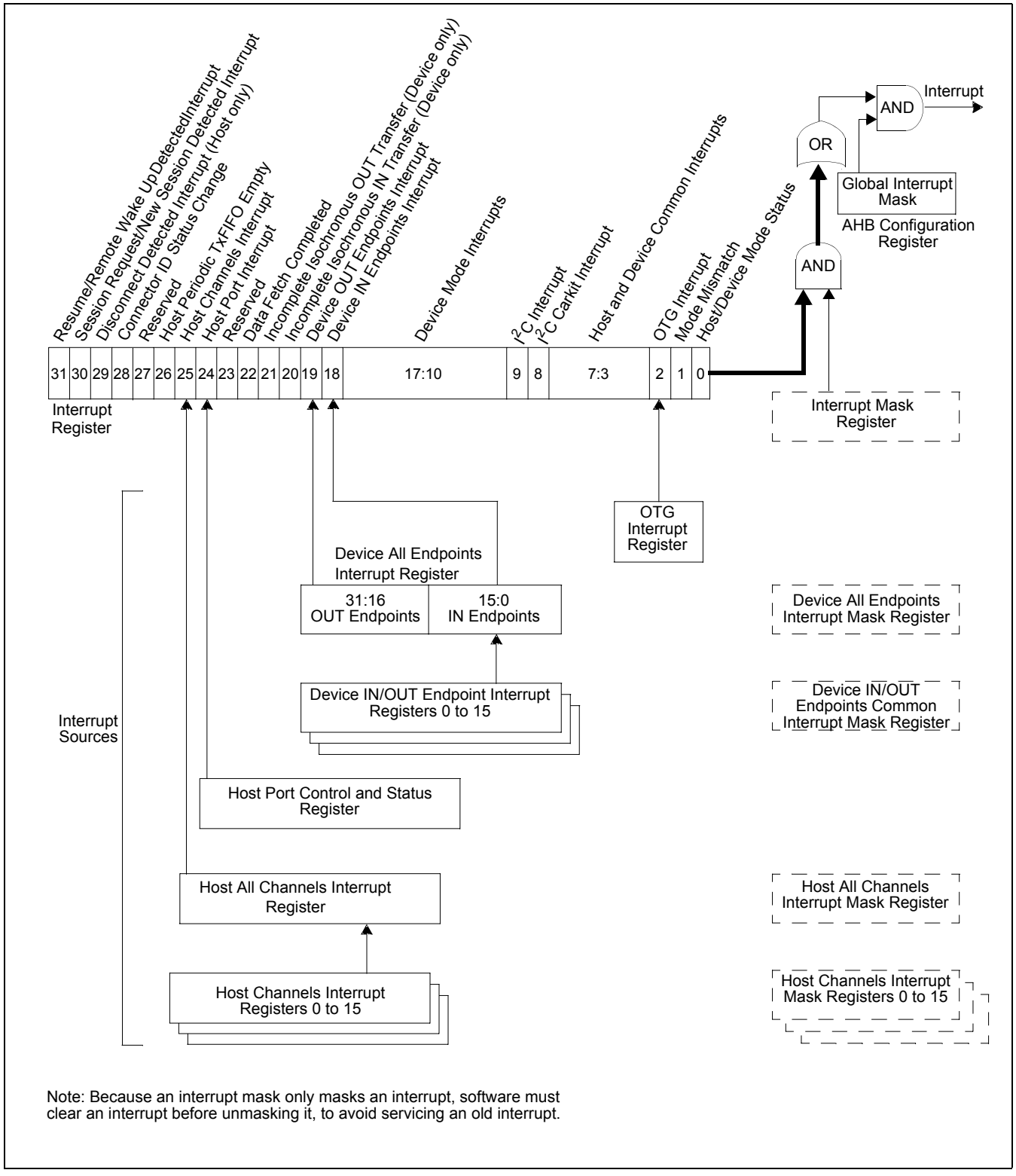
These registers, available in both Host and Device modes, are used to read or write the FIFO space for a specific Endpoint or a channel, in a given direction. If a Host channel is of type IN, the FIFO can only be read on the channel. Similarly, if a Host channel is of type OUT, the FIFO can only be written on the channel. The following table gives the offset from the base address 0x4BFF80000.

Table 38-6. Data FIFO (DFIFO) Access Register Map

FIFO Access Register Section	Address Range	Access
Device IN Endpoint 0/Host OUT Channel 0: DFIFO Write Access Device OUT Endpoint 0/Host IN Channel 0: DFIFO Read Access	0x1000–0x1FFC	WO/RO
Device IN Endpoint 1/Host OUT Channel 1: DFIFO Write Access Device OUT Endpoint 1/Host IN Channel 1: DFIFO Read Access	0x2000–0x2FFC	WO/RO
Device IN Endpoint 2/Host OUT Channel 2: DFIFO Write Access Device OUT Endpoint 2/Host IN Channel 2: DFIFO Read Access	0x3000–0x3FFC	WO/RO

38.4.6 Interrupt Hierarchy

Figure 38-2. Interrupt Hierarchy



Note: Because an interrupt mask only masks an interrupt, software must clear an interrupt before unmasking it, to avoid servicing an old interrupt.

38.5 Register Descriptions

This section describes Global, Device Mode, Host Mode, and Power and Clock Gating CSRs.

38.5.1 Application Access to the CSRs

The Access column of each register description that follows specifies how the application and the function can access the register fields of the CSRs. The following conventions are used:

Read Only (RO)	Register field can only be read by the application. Writes to read-only fields have no effect.
Write Only (WO)	Register field can only be written by the application.
Read and Write (R_W)	Register field can be read and written by the application. The application can set this field by writing 1'b1 and can clear it by writing 1'b0.
Read, Write, and Self Clear (R_W_SC)	Register field can be read and written by the application (Read and Write), and is cleared to 1'b0 by the function (Self Clear). The conditions under which the function clears this field are explained in detail in the field's description.
Read, Write, Self Set, and Self Clear (R_W_SS_SC)	Register field can be read and written by the application (Read and Write), set to 1'b1 by the function on certain USB events (Self Set), and cleared to 1'b0 by the function (Self Clear). The conditions under which the function sets and clears this field are explained in the field's description. (Only the Port Resume bit of the Host Port Control and Status register, HPRT.PrtRes, uses this access type).
Read, Self Set, and Write Clear (R_SS_WC)	Register field can be read by the application (Read), can be set to 1'b1 by the function on a certain internal or USB or AHB event (Self Set), and can be cleared to 1'b0 by the application with a register write of 1'b1 (Write Clear). A register write of 1'b0 has no effect on this field. The conditions under which the function sets this field are explained in detail in the field's description. (For example, interrupt bits.)
Read, Write Set, and Self Clear (R_WS_SC)	Register field can be read by the application (Read), can be set to 1'b1 by the application with a register write of 1'b1 (Write Set), and is cleared to 1'b0 by the function (Self Clear). The application cannot clear this type of field, and a register write of 1'b0 to this bit has no effect on this field. The conditions under which the function clears this field are explained in detail in the field's description. (For example, reset signals)
Read, Self Set, and Self Clear or Write Clear (R_SS_SC_WC)	Register field can be read by the application (Read), can be set to 1'b1 by the function on certain internal or USB or AHB events (Self Set), and can be cleared to 1'b0 either by the function itself (Self Clear) or by the application with a register write of 1'b1 (Write Clear). A register write of 1'b0 to this bit has to no effect on this field. The conditions under which the function sets or clears this field are explained in the field's description. (Only the Port Enable bit of the Host Port Control and Status register, HPRT.PrtEna, and the VStatus Done bit of the PHY Vendor Control register, GPVNDCTL.VStsDone, use this access type.)

Note: Always program Reserved fields with 0s. Treat read values from Reserved fields as unknowns.

38.5.2 Global Registers

These registers are available in both Host and Device modes, and do not need to be reprogrammed when switching between these modes.

38.5.2.1 OTG Control and Status Register (USB0_GOTGCTL)

The OTG Control and Status register controls the behavior and reflects the status of the OTG function of the controller.

Figure 38-3. OTG Control and Status Register (USB0_GOTGCTL)

Field	Description	Mode	Reset	Access
0:11	Reserved		12'h0	
12	B-Session Valid (BSesVld) Indicates the Device mode transceiver status. • 1'b0: B-session is not valid. • 1'b1: B-session is valid.	Device only	1'b0	RO
13	A-Session Valid (ASesVld) Indicates the Host mode transceiver status. • 1'b0: A-session is not valid • 1'b1: A-session is valid	Host only	1'b0	RO
14	Long/Short Debounce Time (DbncTime) Indicates the debounce time of a detected connection. • 1'b0: Long debounce time, used for physical connections (100 ms + 2.5 μs) • 1'b1: Short debounce time, used for soft connections 2.5 μs)	Host only	1'b0	RO
14	Connector ID Status (ConIDSts) Indicates the connector ID status on a connect event. • 1'b0: The USB function is in A-Device mode • 1'b1: The USB function is in B-Device mode	Host and Device	1'b1	RO
16:19	Reserved		4'h0	
20	Device HNP Enabled (DevHNPEn) The application sets this bit when it successfully receives a SetFeature.SetHNPEnable command from the connected USB Host. • 1'b0: HNP is not enabled in the application • 1'b1: HNP is enabled in the application	Device only	1'b0	R_W
21	Host Set HNP Enable (HstSetHNPEn) The application sets this bit when it has successfully enabled HNP (using the SetFeature.SetHNPEnable command) on the connected Device. • 1'b0: Host Set HNP is not enabled • 1'b1: Host Set HNP is enabled	Host only	1'b0	R_W

Figure 38-3. OTG Control and Status Register (USB0_GOTGCTL) (Continued)

Field	Description	Mode	Reset	Access
22	HNP Request (HNPReq) The application sets this bit to initiate an HNP request to the connected USB Host. The application can clear this bit by writing a 0 when the Host Negotiation Success Status Change bit in the OTG Interrupt register (GOTGINT.HstNegSucStsChng) is set. The function clears this bit when the HstNegSucStsChng bit is cleared. • 1'b0: No HNP request • 1'b1: HNP request	Device only	1'b0	R_W
23	Host Negotiation Success (HstNegScs) The function sets this bit when Host negotiation is successful. The function clears this bit when the HNP Request (HNPReq) bit in this register is set. • 1'b0: Host negotiation failure • 1'b1: Host negotiation success	Device only	1'b0	RO
24:29	Reserved		4'h0	
30	Session Request (SesReq) The application sets this bit to initiate a session request on the USB. The application can clear this bit by writing a 0 when the Host Negotiation Success Status Change bit in the OTG Interrupt register (GOTGINT.HstNegSucStsChng) is set. The function clears this bit when the HstNegSucStsChng bit is cleared. If you use the USB 1.1 Full-Speed Serial Transceiver interface to initiate the session request, the application must wait until the VBUS discharges to 0.2 V, after the B-Session Valid bit in this register (GOTGCTL.BSesVld) is cleared. This discharge time varies between different PHYs and can be obtained from the PHY vendor. • 1'b0: No session request • 1'b1: Session request	Device only	1'b0	R_W
31	Session Request Success (SesReqScs) The function sets this bit when a session request initiation is successful. • 1'b0: Session request failure • 1'b1: Session request success	Device only	1'b0	RO

38.5.2.2 OTG Interrupt Register (USB0_GOTGINT)

The application reads this register whenever there is an OTG interrupt and clears the bits in this register to clear the OTG interrupt.

Figure 38-4. OTG Interrupt Register (USB0_GOTGINT)

Field	Description	Mode	Reset	Access
0:11	Reserved		12'h0	

Figure 38-4. OTG Interrupt Register (USB0_GOTGINT) (Continued)

Field	Description	Mode	Reset	Access
12	Debounce Done (DbnceDone) The function sets this bit when the debounce is completed after the Device connect. The application can start driving USB reset after seeing this interrupt. This bit is only valid when the HNP Capable or SRP Capable bit is set in the function USB Configuration register (GUSBCFG.HNPCap or GUSBCFG.SRPCap, respectively).	Host and Device	1'b0	R_SS_WC
13	A-Device Timeout Change (ADevTOUTChg) The function sets this bit to indicate that the A-Device has timed out while waiting for the B-Device to connect.	Host and Device	1'b0	R_SS_WC
14	Host Negotiation Detected (HstNegDet) The function sets this bit when it detects a Host negotiation request on the USB.	Host and Device	1'b0	R_SS_WC
15:21	Reserved		6'h0	
22	Host Negotiation Success Status Change (HstNegSucStsChng) The function sets this bit on the success or failure of a USB Host negotiation request. The application must read the Host Negotiation Success bit of the OTG Control and Status register (GOTGCTL.HstNegScs) to check for success or failure.	Host and Device	1'b0	R_SS_WC
23	Session Request Success Status Change (SesReqSucStsChng) The function sets this bit on the success or failure of a session request. The application must read the Session Request Success bit in the OTG Control and Status register (GOTGCTL.SesReqScs) to check for success or failure.	Host and Device	1'b0	R_SS_WC
24:28	Reserved		6'h0	
29	Session End Detected (SesEndDet) The function sets this bit when the utmiotg_bvalid signal is deasserted.	Host and Device	1'b0	R_SS_WC
30:31	Reserved		2'h0	

38.5.2.3 AHB Configuration Register (USB0_GAHBCFG)

This register can be used to configure the function after power-on or a change in mode of operation. This register mainly contains AHB system-related configuration parameters. Do not change this register after the initial programming. The application must program this register before starting any transactions on either the AHB or the USB.

Figure 38-5. AHB Configuration Register (USB0_GAHBCFG)

Field	Description	Mode	Reset	Access
0:22	Reserved		22'h0	
23	Periodic TxFIFO Empty Level (PTxFEmpLvl) Indicates when the Periodic TxFIFO Empty Interrupt bit in the function Interrupt register (GINTSTS.PTxFEmp) is triggered. This bit is used only in Slave mode. • 1'b0: GINTSTS.PTxFEmp interrupt indicates that the Periodic TxFIFO is half empty • 1'b1: GINTSTS.PTxFEmp interrupt indicates that the Periodic TxFIFO is completely empty	Host only	1'b0	R_W
24	Non-Periodic TxFIFO Empty Level (NPTxFEmpLvl) This bit is used only in Slave mode. In Host mode and with Shared FIFO with Device mode, this bit indicates when the Non-Periodic TxFIFO Empty Interrupt bit in the function Interrupt register (GINTSTS.NPTxFEmp) is triggered. With dedicated FIFO in Device mode, this bit indicates when IN Endpoint Transmit FIFO empty interrupt (DIEPINTn.TxFEmp) is triggered. Host mode and with Shared FIFO with Device mode: • 1'b0: GINTSTS.NPTxFEmp interrupt indicates that the Non-Periodic TxFIFO is half empty • 1'b1: GINTSTS.NPTxFEmp interrupt indicates that the Non-Periodic TxFIFO is completely empty • Dedicated FIFO in Device mode: • 1'b0: DIEPINTn.TxFEmp interrupt indicates that the IN Endpoint TxFIFO is half empty • 1'b1: DIEPINTn.TxFEmp interrupt indicates that the IN Endpoint TxFIFO is completely empty	Host and Device	1'b0	R_W
25	Reserved			
26	DMA Enable (DMAEn) • 1'b0: Function operates in Slave mode • 1'b1: Function operates in a DMA mode	Host and Device	1'b0	R_W
27:30	Burst Length/Type (HBstLen) This field is used in Internal DMA mode to take the DMA command and split the data transfer into multiples of AHB bursts based on the burst type. Can support INCR, INCR4, INCR8, INCR16 commands. Always break on the burst size boundary, thus also satisfying the AHB requirement of no 1K boundary crossing. Internal DMA Mode—AHB Master burst type: • 4'b0000 Single Single DWORD transfer • 4'b0001 INCR Burst break on the 1K bytes boundary • 4'b0011 INCR4 Burst break on the 16 bytes boundary • 4'b0101 INCR8 Burst break on the 32 bytes boundary • 4'b0111 INCR16 Burst break on the 64 bytes boundary • Others: Reserved Do one transfer at a time. No concurrency allowed.	Host and Device	4'b0	R_W

Figure 38-5. AHB Configuration Register (USB0_GAHBCFG) (Continued)

Field	Description	Mode	Reset	Access
31	Global Interrupt Mask (GlblIntrMsk) The application uses this bit to mask or unmask the interrupt line assertion to itself. Irrespective of this bit's setting, the interrupt status registers are updated by the function. • 1'b0: Mask the interrupt assertion to the application. • 1'b1: Unmask the interrupt assertion to the application.	Host and Device	1'b0	R_W

38.5.2.4 USB Configuration Register (USB0_GUSBCFG)

This register can be used to configure the function after power-on or a changing to Host mode or Device mode. It contains USB and USB-PHY related configuration parameters. The application must program this register before starting any transactions on either the AHB or the USB. Do not make changes to this register after the initial programming.

Figure 38-6. USB Configuration Register (USB0_GUSBCFG)

Field	Description	Mode	Reset	Access
0	Corrupt Tx packet This bit is for debug purposes only. Never set this bit to 1.	Host and Device	1'b0	R_W
1	Force Device Mode (ForceDevMode) Writing a 1 to this bit will force the function to Device mode irrespective of utmiotg_iddig input pin. • 1'b0 : Normal Mode. • 1'b1 : Force Device Mode.	Host and Device	1'b0	R_W
2	Force Host Mode (ForceHstMode) Writing a 1 to this bit will force the function to Host mode irrespective of utmiotg_iddig input pin. • 1'b0 : Normal Mode. • 1'b1 : Force Host Mode.	Host and Device	1'b0	R_W
3:17	Reserved		0	
18:21	USB Turnaround Time (USBTrdTim) Sets the turnaround time in PHY clocks. Specifies the response time for a MAC request to the Packet FIFO Controller (PFC) to fetch data from the DFIFO (SPRAM). This must be programmed to: • 4'h5: When the MAC interface is 16-bit UTMI+. • 4'h9: When the MAC interface is 8-bit UTMI+. Note: If USB turnaround time is not critical, these bits can be programmed to a larger value.	Device only	4'h5	R_W
22	HNP-Capable (HNPCap) The application uses this bit to control the USB function HNP capabilities. • 1'b0: HNP capability is not enabled. • 1'b1: HNP capability is enabled.	Host and Device	1'b0	RO/R_W

Figure 38-6. USB Configuration Register (USB0_GUSBCFG) (Continued)

Field	Description	Mode	Reset	Access
23	SRP-Capable (SRPCap) The application uses this bit to control the USB function SRP capabilities. If the function operates as a non-SRP-capable B-Device, it cannot request the connected A-Device (Host) to activate V BUS and start a session. • 1'b0: SRP capability is not enabled. • 1'b1: SRP capability is enabled. This bit is writable only if an SRP mode was specified for Mode of Operation in functionConsultant (parameter OTG_MODE). Otherwise, reads return 0.	Host and Device	1'b1	RO/R_W
24	Reserved (Must be 0)		0	
25	USB 2.0 High-Speed PHY or USB 1.1 Full-Speed Serial Transceiver Select (PHYSel) The application uses this bit to select a high-speed ULPI PHY, or a full-speed transceiver. • 1'b0: USB 2.0 high-speed ULPI PHY • 1'b1: USB 1.1 full-speed serial transceiver	Host and Device	1'b0	WO/R_W
26	Reserved (Must be 0)		0	na
27	Reserved (Default is 1)		1	na
28	Reserved		0	
29:31	HS/FS Timeout Calibration (TOutCal) The number of PHY clocks that the application programs in this field is added to the high-speed/full-speed interpacket timeout duration in the function to account for any additional delays introduced by the PHY. This can be required, because the delay introduced by the PHY in generating the line state condition can vary from one PHY to another. The USB standard timeout value for high-speed operation is 736 to 816 (inclusive) bit times. The USB standard timeout value for full-speed operation is 16 to 18 (inclusive) bit times. The application must program this field based on the speed of enumeration. The number of bit times added per PHY clock are: High-speed operation: • One 30-MHz PHY clock = 16 bit times • One 60-MHz PHY clock = 8 bit times Full-speed operation: • One 30-MHz PHY clock = 0.4 bit times • One 60-MHz PHY clock = 0.2 bit times • One 48-MHz PHY clock = 0.25 bit times	Host and Device	3'h0	R_W

38.5.2.5 Reset Register (USB0_GRSTCTL)

The application uses this register to reset various hardware features inside the function.

Figure 38-7. Reset Register (USB0_GRSTCTL)

Field	Description	Mode	Reset	Access
0	AHB Master Idle (AHBIdle) Indicates that the AHB Master State Machine is in the IDLE condition.	Host and Device	1'b1	RO
1	DMA Request Signal (DMAReq) Indicates that the DMA request is in progress. Used for debug.	Host and Device	1'b0	RO
2:20	Reserved			
21:25	TxFIFO Number (TxFNum) This is the FIFO number that must be flushed using the TxFIFO Flush bit. This field must not be changed until the function clears the TxFIFO Flush bit. • 5'h0: - Non-periodic TxFIFO flush in Host mode - Non-periodic TxFIFO flush in Device mode when in shared FIFO operation - Tx FIFO 0 flush in Device mode when in dedicated FIFO mode • 5'h1: - Periodic TxFIFO flush in Host mode - Periodic TxFIFO 1 flush in Device mode when in shared FIFO operation - TXFIFO 1 flush in Device mode when in dedicated FIFO mode • 5'h2: - Periodic TxFIFO 2 flush in Device mode when in shared FIFO operation - TXFIFO 2 flush in Device mode when in dedicated FIFO mode ... • 5'hF: - Periodic TxFIFO 15 flush in Device mode when in shared FIFO operation - TXFIFO 15 flush in Device mode when in dedicated FIFO mode • 5'h10: Flush all the transmit FIFOs in Device or Host mode.	Host and Device	5'h0	R_W
26	TxFIFO Flush (TxFFlsh) This bit selectively flushes a single or all transmit FIFOs, but cannot do so if the function is in the midst of a transaction. The application must write this bit only after checking that the function is neither writing to the TxFIFO nor reading from the TxFIFO. Verify using these registers: • Read—NAK Effective Interrupt ensures the function is not reading from the FIFO • Write—GRSTCTL.AHBIdle ensures the function is not writing anything to the FIFO. Flushing is normally recommended when FIFOs are reconfigured or when switching between Shared FIFO and Dedicated Transmit FIFO operation. FIFO flushing is also recommended during Device Endpoint disable. The application must wait until the function clears this bit before performing any operations. This bit takes eight clocks to clear, using the slower clock of phy_clk or hclk.	Host and Device	1'b0	R_WS_SC

Figure 38-7. Reset Register (USB0_GRSTCTL) (Continued)

Field	Description	Mode	Reset	Access
27	RxFIFO Flush (RxFFlsh) The application can flush the entire RxFIFO using this bit, but must first ensure that the function is not in the middle of a transaction. The application must only write to this bit after checking that the function is neither reading from the RxFIFO nor writing to the RxFIFO. The application must wait until the bit is cleared before performing any other operations. This bit requires eight clocks (slowest of PHY or AHB clock) to clear.	Host and Device	1'b0	R_WS_SC
28	IN Token Sequence Learning Queue Flush (INTknQFlsh) The application writes this bit to flush the IN Token Sequence Learning Queue.	Device only	1'b0	R_WS_SC
29	Host Frame Counter Reset (FrmCntrRst) The application writes this bit to reset the (micro)frame number counter inside the function. When the (micro)frame counter is reset, the subsequent SOF sent out by the function has a (micro)frame number of 0.	Host only	1'b0	R_WS_SC
30	HClk Soft Reset (HSftRst) The application uses this bit to flush the control logic in the AHB Clock domain. Only AHB Clock Domain pipelines are reset. • FIFOs are not flushed with this bit. • All state machines in the AHB clock domain are reset to the Idle state after terminating the transactions on the AHB, following the protocol. • CSR control bits used by the AHB clock domain state machines are cleared. • To clear this interrupt, status mask bits that control the interrupt status and are generated by the AHB clock domain state machine are cleared. • Because interrupt status bits are not cleared, the application can get the status of any function events that occurred after it set this bit. This is a self-clearing bit that the function clears after all necessary logic is reset in the function. This can take several clocks, depending on the function's current state.	Host and Device	1'b0	R_WS_SC

Figure 38-7. Reset Register (USB0_GRSTCTL) (Continued)

Field	Description	Mode	Reset	Access
31	Soft Reset (CSftRst) Resets the hclk and phy_clock domains as follows: • Clears the interrupts and all the CSR registers except the following register bits: - PCGCCTL.GateHclk - PCGCCTL.StopPclk - GUSBCFG.PhyLPwrClkSel - GUSBCFG.DDRSel - GUSBCFG.PHYSel - GUSBCFG.FSIntf - GUSBCFG.ULPI_UTMI_Sel - GUSBCFG.PHYIf - HCFG.FSLSPclkSel - DCFG.DevSpd - GGPIO • All module state machines (except the AHB Slave Unit) are reset to the IDLE state, and all the transmit FIFOs and the receive FIFO are flushed. • Any transactions on the AHB Master are terminated as soon as possible, after gracefully completing the last data phase of an AHB transfer. Any transactions on the USB are terminated immediately. The application can write to this bit any time it wants to reset the function. This is a self-clearing bit and the function clears this bit after all the necessary logic is reset in the function, which can take several clocks, depending on the current state of the function. Once this bit is cleared software must wait at least three PHY clocks before doing any access to the PHY domain (synchronization delay). Software must also must check that AHBIdle of this register is 1 (AHB Master is IDLE) before starting any operation. Typically software reset is used during software development and also when you dynamically change the PHY selection bits in the USB configuration registers listed above. When you change the PHY, the corresponding clock for the PHY is selected and used in the PHY domain. Once a new clock is selected, the PHY domain has to be reset for proper operation.	Host and Device	1'b0	R_WS_SC

38.5.2.6 Interrupt Register (USB0_GINTSTS)

This register interrupts the application for system-level events in the current mode of operation (Device mode or Host mode).

Some of the bits in this register are valid only in Host mode, while others are valid in Device mode only. This register also indicates the current mode of operation. In order to clear the interrupt status bits of type R_SS_WC, the application must write 1'b1 into the bit.

The FIFO status interrupts are read only; once software reads from or writes to the FIFO while servicing these interrupts, FIFO interrupt conditions are cleared automatically.

Figure 38-8. Interrupt Register (USB0_GINTSTS)

Field	Description	Mode	Reset	Access
0	Resume/Remote Wake Up Detected Interrupt (WkUpInt) In Device mode, this interrupt is asserted when a resume is detected on the USB. In Host mode, this interrupt is asserted when a remote wake up is detected on the USB.	Host and Device	1'b0	R_SS_WC
1	Session Request/New Session Detected Interrupt (SessReqInt) In Host mode, this interrupt is asserted when a session request is detected from the Device. In Device mode, this interrupt is asserted when the utmiotg_bvalid signal goes high.	Host and Device	1'b0	R_SS_WC
2	Disconnect Detected Interrupt (DisconnInt) Asserted when a Device disconnect is detected.	Host and Device	1'b0	R_SS_WC
3	Connector ID Status Change (ConIDStsChng) The function sets this bit when there is a change in connector ID status.	Host and Device	1'b0	R_SS_WC
4	Reserved		1'b0	
5	Periodic TxFIFO Empty (PTxFEmp) Asserted when the Periodic Transmit FIFO is either half or completely empty and there is space for at least one entry to be written in the Periodic Request Queue. The half or completely empty status is determined by the Periodic TxFIFO Empty Level bit in the AHB Configuration register (GAHBCFG.PTxFEmpLvl).	Host only	1'b0	RO
6	Host Channels Interrupt (HChInt) The function sets this bit to indicate that an interrupt is pending on one of the channels of the function (in Host mode). The application must read the Host All Channels Interrupt (HAINT) register to determine the exact number of the channel on which the interrupt occurred, and then read the corresponding Host Channel-n Interrupt (HCINTn) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the HCINTn register to clear this bit.	Host only	1'b0	RO
7	Host Port Interrupt (PrtInt) The function sets this bit to indicate a change in port status of one of the USB function ports in Host mode. The application must read the Host Port Control and Status (HPRT) register to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in the Host Port Control and Status register to clear this bit.	Host only	1'b0	RO
8	Reserved			

Figure 38-8. Interrupt Register (USB0_GINTSTS) (Continued)

Field	Description	Mode	Reset	Access
9	Data Fetch Suspended (FetSusp) This interrupt is valid only in DMA mode. This interrupt indicates that the function has stopped fetching data for IN endpoints due t the unavailability of TxFIFO space or Request Queue space. This interrupt is used by the application for an Endpoint mismatch algorithm. For example, after detecting an Endpoint mismatch, the application: • Sets a global non-periodic IN NAK handshake • Disables In endpoints • Flushes the FIFO • Determines the token sequence from the IN Token Sequence Learning Queue • Re-enables the endpoints • Clears the global non-periodic IN NAK handshake If the global non-periodic IN NAK is cleared, the function has not yet fetched data for the IN Endpoint, and the IN token is received: the function generates an "IN token received when FIFO empty" interrupt. The OTG then sends the Host a NAK response. To avoid this scenario, the application can check the GINTSTS.FetSusp interrupt, which ensures that the FIFO is full before clearing a global NAK handshake. Alternatively, the application can mask the "IN token received when FIFO empty" interrupt when clearing a global IN NAK handshake.	Device only	1'b0	R_SS_WC
10	Incomplete Periodic Transfer (incomplP) In Host mode, the function sets this interrupt bit when there are incomplete periodic transactions still pending which are scheduled for the current microframe. Incomplete Isochronous OUT Transfer (incompISOOUT) The Device mode, the function sets this interrupt to indicate that there is at least one isochronous OUT Endpoint on which the transfer is not completed in the current microframe. This interrupt is asserted along with the End of Periodic Frame Interrupt (EOPF) bit in this register.	Host only Device only	1'b0	R_SS_WC
11	Incomplete Isochronous IN Transfer (incompISOIN) The function sets this interrupt to indicate that there is at least one isochronous IN Endpoint on which the transfer is not completed in the current microframe. This interrupt is asserted along with the End of Periodic Frame Interrupt (EOPF) bit in this register.	Device only	1'b0	R_SS_WC
12	OUT Endpoints Interrupt (OEPInt) The function sets this bit to indicate that an interrupt is pending on one of the OUT endpoints of the function (in Device mode). The application must read the Device All Endpoints Interrupt (DAINT) register to determine the exact number of the OUT Endpoint on which the interrupt occurred, and then read the corresponding Device OUT Endpoint-n Interrupt (DOEPINTn) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding DOEPINTn register to clear this bit.	Device only	1'b0	RO

Figure 38-8. Interrupt Register (USB0_GINTSTS) (Continued)

Field	Description	Mode	Reset	Access
13	IN Endpoints Interrupt (IEPInt) The function sets this bit to indicate that an interrupt is pending on one of the IN endpoints of the function (in Device mode). The application must read the Device All Endpoints Interrupt (DAINT) register to determine the exact number of the IN Endpoint on which the interrupt occurred, and then read the corresponding Device IN Endpoint-n Interrupt (DIEPINTn) register to determine the exact cause of the interrupt. The application must clear the appropriate status bit in the corresponding DIEPINTn register to clear this bit.	Device only	1'b0	RO
14	Endpoint Mismatch Interrupt (EPMis) Note: This interrupt is valid only in shared FIFO operation. Indicates that an IN token has been received for a non-periodic Endpoint, but the data for another Endpoint is present in the top of the Non-periodic Transmit FIFO and the IN Endpoint mismatch count programmed by the application has expired.	Device only	1'b0	RO
15	Reserved		1'b0	
16	End of Periodic Frame Interrupt (EOPF) Indicates that the period specified in the Periodic Frame Interval field of the Device Configuration register (DCFG.PerFrInt) has been reached in the current microframe.	Device only	1'b0	R_SS_WC
17	Isochronous OUT Packet Dropped Interrupt (ISOOutDrop) The function sets this bit when it fails to write an isochronous OUT packet into the RxFIFO because the RxFIFO doesn't have enough space to accommodate a maximum packet size packet for the isochronous OUT Endpoint.	Device only	1'b0	R_SS_WC
18	Enumeration Done (EnumDone) The function sets this bit to indicate that speed enumeration is complete. The application must read the Device Status (DSTS) register to obtain the enumerated speed.	Device only	1'b0	R_SS_WC
19	USB Reset (USBRst) The function sets this bit to indicate that a reset is detected on the USB.	Device only	1'b0	R_SS_WC
20	USB Suspend (USBSusp) The function sets this bit to indicate that a suspend was detected on the USB. The function enters the Suspended state when there is no activity on the phy_line_state_i signal for an extended period of time.	Device only	1'b0	R_SS_WC
21	Early Suspend (ErlySusp) The function sets this bit to indicate that an Idle state has been detected on the USB for 3 ms.	Device only	1'b0	R_SS_WC
22:23	Reserved		0	
24	Global OUT NAK Effective (GOUTNakEff) Indicates that the Set Global OUT NAK bit in the Device Control register (DCTL.SGOUTNak), set by the application, has taken effect in the function. This bit can be cleared by writing the Clear Global OUT NAK bit in the Device Control register (DCTL.CGOUTNak).	Device only	1'b0	RO

Figure 38-8. Interrupt Register (USB0_GINTSTS) (Continued)

Field	Description	Mode	Reset	Access
25	Global IN Non-periodic NAK Effective (GINNakEff) Indicates that the Set Global Non-periodic IN NAK bit in the Device Control register (DCTL.SGNPInNak), set by the application, has taken effect in the function. That is, the function has sampled the Global IN NAK bit set by the application. This bit can be cleared by clearing the Clear Global Non-periodic IN NAK bit in the Device Control register (DCTL.CGNPInNak). This interrupt does not necessarily mean that a NAK handshake is sent out on the USB. The STALL bit takes precedence over the NAK bit.	Device only	1'b0	RO
26	Non-periodic TxFIFO Empty (NPTxFEmp) This interrupt is asserted when the Non-periodic TxFIFO is either half or completely empty, and there is space for at least one entry to be written to the Non-periodic Transmit Request Queue. The half or completely empty status is determined by the Non-periodic TxFIFO Empty Level bit in the function AHB Configuration register (GAHBCFG.NPTxFEmpLvl).	Host and Device	1'b0	RO
27	RxFIFO Non-Empty (RxFLvl) Indicates that there is at least one packet pending to be read from the RxFIFO.	Host and Device	1'b0	RO
28	Start of (micro)Frame (SOF) In Host mode, the function sets this bit to indicate that an SOF (FS), micro-SOF (HS), or Keep-Alive (LS) is transmitted on the USB. The application must write a 1 to this bit to clear the interrupt. In Device mode, in the function sets this bit to indicate that an SOF token has been received on the USB. The application can read the Device Status register to get the current (micro)frame number. This interrupt is seen only when the function is operating at either HS or FS.	Host and Device	1'b0	R_SS_WC
29	OTG Interrupt (OTGInt) The function sets this bit to indicate an OTG protocol event. The application must read the OTG Interrupt Status (GOTGINT) register to determine the exact event that caused this interrupt. The application must clear the appropriate status bit in the GOTGINT register to clear this bit.	Host and Device	1'b0	RO
30	Mode Mismatch Interrupt (ModeMis) The function sets this bit when the application is trying to access: • A Host mode register, when the function is operating in Device mode • A Device mode register, when the function is operating in Host mode The register access is completed on the AHB with an OKAY response, but is ignored by the function internally and doesn't affect the operation of the function.	Host and Device	1'b0	R_SS_WC
31	Current Mode of Operation (CurMod) Indicates the current mode of operation. • 1'b0: Device mode • 1'b1: Host mode	Host and Device	1'b0	RO

38.5.2.7 Interrupt Mask Register (USB0_GINTMSK)

This register works with the Interrupt register to interrupt the application. When an interrupt bit is masked, the interrupt associated with that bit is not generated. However, the Interrupt (GINTSTS) register bit corresponding to that interrupt is still set.

- Mask interrupt: 1'b0
- Unmask interrupt: 1'b1

Figure 38-9. Interrupt Mask Register (USB0_GINTMSK)

Field	Description	Mode	Reset	Access
0	Resume/Remote Wake Up Detected Interrupt Mask (WkUpIntMsk)	Host and Device	1'b0	R_W
1	Session Request/New Session Detected Interrupt Mask (SessReqIntMsk)	Host and Device	1'b0	R_W
2	Disconnect Detected Interrupt Mask (DisconnIntMsk)	Host and Device	1'b0	R_W
3	Connector ID Status Change Mask (ConIDStsChngMsk)	Host and Device	1'b0	R_W
4	Reserved		1'b0	
5	Periodic TxFIFO Empty Mask (PTxFEmpMsk)	Host only	1'b0	R_W
6	Host Channels Interrupt Mask (HChIntMsk)	Host only	1'b0	R_W
7	Host Port Interrupt Mask (PrtIntMsk)	Host only	1'b0	R_W
8	Reserved		1'b0	
9	Data Fetch Suspended Mask (FetSuspMsk)	Device only	1'b0	R_W
10	Incomplete Periodic Transfer Mask (incomplPMsk) Incomplete Isochronous OUT Transfer Mask (incompISOOUTMsk)	Host only Device only	1'b0	R_W
11	Incomplete Isochronous IN Transfer Mask (incompISOINMsk)	Device only	1'b0	R_W
12	OUT Endpoints Interrupt Mask (OEPIntMsk)	Device only	1'b0	R_W
13	IN Endpoints Interrupt Mask (INEPIntMsk)	Device only	1'b0	R_W
14	Endpoint Mismatch Interrupt Mask (EPMisMsk)	Device only	1'b0	R_W
15	Reserved		1'b0	
16	End of Periodic Frame Interrupt Mask (EOPFMsk)	Device only	1'b0	R_W
17	Isochronous OUT Packet Dropped Interrupt Mask (ISOOutDropMsk)	Device only	1'b0	R_W
18	Enumeration Done Mask (EnumDoneMsk)	Device only	1'b0	R_W
19	USB Reset Mask (USBRstMsk)	Device only	1'b0	R_W
20	USB Suspend Mask (USBSuspMsk)	Device only	1'b0	R_W
21	Early Suspend Mask (ErlySuspMsk)	Device only	1'b0	R_W
22:23	Reserved		0	
24	Global OUT NAK Effective Mask (GOUTNakEffMsk)	Device only	1'b0	R_W

Figure 38-9. Interrupt Mask Register (USB0_GINTMSK) (Continued)

Field	Description	Mode	Reset	Access
25	Global Non-periodic IN NAK Effective Mask (GINNakEffMsk)	Device only	1'b0	R_W
26	Non-periodic TxFIFO Empty Mask (NPTxFEmpMsk)	Host and Device	1'b0	R_W
27	Receive FIFO Non-Empty Mask (RxFLvlMsk)	Host and Device	1'b0	R_W
28	Start of (micro)Frame Mask (SofMsk)	Host and Device	1'b0	R_W
29	OTG Interrupt Mask (OTGIntMsk)	Host and Device	1'b0	R_W
30	Mode Mismatch Interrupt Mask (ModeMisMsk)	Host and Device	1'b0	R_W
31	Reserved		1'b0	

38.5.2.8 Rx Status Debug Read/Status Read/Pop Registers (USB0_GRXSTSR/GRXSTSP)

Offset for Read: 01Ch

Offset for Pop: 020h

A read to the Receive Status Debug Read register returns the contents of the top of the Receive FIFO. A read to the Receive Status Read and Pop register additionally pops the top data entry out of the RxFIFO.

The receive status contents must be interpreted differently in Host and Device modes. The function ignores the receive status pop/read when the receive FIFO is empty and returns a value of 32'h0000_0000. The application must only pop the Receive Status FIFO when the Receive FIFO Non-Empty bit of the Interrupt register (GINTSTS.RxFLvl) is asserted.

Figure 38-10 shows the use of these registers in Host mode, and *Figure 38-11* shows Device mode.

Figure 38-10. HM Rx Status Debug Read/Status Read and Pop Reg (USB0_GRXSTSR/GRXSTSP)

Field	Description	Reset	Access
0:10	Reserved 11'h0	11'h0	
11:14	Packet Status (PktSts) Indicates the status of the received packet • 4'b0010: IN data packet received • 4'b0011: IN transfer completed (triggers an interrupt) • 4'b0101: Data toggle error (triggers an interrupt) • 4'b0111: Channel halted (triggers an interrupt) • Others: Reserved	4'b0	RO
15:16	Data PID (DPID) Indicates the Data PID of the received packet • 2'b00: DATA0 • 2'b10: DATA1 • 2'b01: DATA2 • 2'b11: MDATA	2'b0	RO
17:27	Byte Count (BCnt) Indicates the byte count of the received IN data packet.	11'h0	RO
28:31	Channel Number (ChNum) Indicates the channel number to which the current received packet belongs.	4'h0	RO

Figure 38-11. DM Rx Status Debug Read/Status Read and Pop Reg (USB0_GRXSTSR/GRXSTSP)

Field	Description	Reset	Access
0:6	Reserved	7'h0	
7:10	Frame Number (FN) This is the least significant four bits of the (micro)frame number in which the packet is received on the USB. This field is supported only when isochronous OUT endpoints are supported.	4'h0	RO
11:14	Packet Status (PktSts) Indicates the status of the received packet • 4'b0001: Global OUT NAK (triggers an interrupt) • 4'b0010: OUT data packet received • 4'b0011: OUT transfer completed (triggers an interrupt) • 4'b0100: SETUP transaction completed (triggers an interrupt) • 4'b0110: SETUP data packet received • Others: Reserved	4'h0	RO
15:16	Data PID (DPID) Indicates the Data PID of the received OUT data packet • 2'b00: DATA0 • 2'b10: DATA1 • 2'b01: DATA2 • 2'b11: MDATA	2'b0	RO
17:27	Byte Count (BCnt) Indicates the byte count of the received data packet.	11'h0	RO
28:31	Endpoint Number (EPNum) Indicates the Endpoint number to which the current received packet belongs.	4'h0	RO

38.5.2.9 Receive FIFO Size Register (USB0_GRXFSIZ)

The application can program the RAM size that must be allocated to the RxFIFO.

Figure 38-12. Receive FIFO Size Register (USB0_GRXFSIZ)

Field	Description	Reset	Access
0:15	Reserved	16'h0	
16:31	RxFIFO Depth (RxFDep) This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 32,768 The power-on reset value of this register is specified as the Largest Rx Data FIFO Depth = 531. You can write a new value in this field. Programmed values must not exceed the default value.	User selected	RO/R_W

38.5.2.10 Non-Periodic Transmit FIFO Size Register (USB0_GNPTXFSIZ)

The application can program the RAM size and the memory start address for the Non-periodic TxFIFO.

Figure 38-13. Non-Periodic Transmit FIFO Size Register (USB0_GNPTXFSIZ)

Field	Description	Reset	Access
0:15	Non-periodic TxFIFO Depth (NPTxFDep) Valid for both Host and Device mode. This value is in terms of 32-bit words. Minimum value is 16 Maximum value is 32,768 The power-on reset value of this register is specified as the Largest Non-periodic Tx Data FIFO Depth = 256. You can write a new value in this field. Programmed values must not exceed the power-on value.	User selected	RO/R_W
16:31	Non-periodic Transmit RAM Start Address Valid for both Host and Device mode. The power-on reset value of this register is specified as the Largest Non-periodic Tx Data FIFO Depth = 531. You can write that value in this field. Programmed values must not exceed the power-on value.	User selected	RO/R_W

38.5.2.11 Non-Periodic Transmit FIFO/Queue Status Register (USB0_GNPTXSTS)

In Device mode, this register is valid only in Shared FIFO operation.

This read-only register contains the free space information for the Non-periodic TxFIFO and the Non-periodic Transmit Request Queue.

.

Figure 38-14. Non-Periodic Transmit FIFO/Queue Status Register (USB0_GNPTXSTS)

Field	Description	Reset	Access
0	Reserved	1'b0	
1:7	Top of the Non-periodic Transmit Request Queue (NPTxQTop) Entry in the Non-periodic Tx Request Queue that is currently being processed by the MAC. • Bits 1:4: Channel/Endpoint number • Bits 5:6: - 2'b00: IN/OUT token - 2'b01: Zero-length transmit packet (Device IN/Host OUT) - 2'b10: PING/CSPLIT token - 2'b11: Channel halt command • Bit 7: Terminate (last entry for selected channel/Endpoint)	7'h0	RO
8:15	Non-periodic Transmit Request Queue Space Available (NPTxQSpcAvail) Indicates the amount of free space available in the Non-periodic Transmit Request Queue. This queue holds both IN and OUT requests in Host mode. Device mode has only IN requests. • 8'h0: Non-periodic Transmit Request Queue is full • 8'h1: 1 location available • 8'h2: 2 locations available • n: n locations available ($0 \leq n \leq 8$) • Others: Reserved	User selected	RO
16:31	Non-periodic TxFIFO Space Avail (NPTxFSpcAvail) Indicates the amount of free space available in the Non-periodic TxFIFO. Values are in terms of 32-bit words. • 16'h0: Non-periodic TxFIFO is full • 16'h1: 1 word available • 16'h2: 2 words available • 16'hn: n words available (where $0 \leq n \leq 32,768$) • 16'h8000: 32,768 words available • Others: Reserved	User selected	RO

38.5.2.12 PHY Vendor Control Register (USB0_GPVNDCTL)

The application can use this register to access PHY registers.

For a ULPI PHY, the function uses the ULPI interface for PHY register access. The application sets Vendor Control register for PHY register access and times the PHY register access. The application polls the VStatus Done bit in this register for the completion of the PHY register access.

:

Figure 38-15. PHY Vendor Control Register (USB0_GPVNDCTL)

Field	Description	Reset	Access
0	Disable ULPI Drivers (DisUlpiDrvr) This field is used only if the Carkit interface was enabled in functionConsultant (parameter OTG_ULPI_CARKIT = 1). Otherwise, reads return 0. Software sets this bit when it has finished processing the ULPI Carkit Interrupt (GINTSTS.ULPICKINT). When set, the USB function disables drivers for output signals and masks input signal for the ULPI interface. The USB function clears this bit before enabling the ULPI interface.	1'b0	R_WS_SC
1:3	Reserved	3'b0	
4	VStatus Done (VStsDone) The function sets this bit when the vendor control access is done. This bit is cleared by the function when the application sets the New Register Request bit.	1'b0	R_SS_WC_SC
5	VStatus Busy (VStsBsy) The function sets this bit when the vendor control access is in progress and clears this bit when done.	1'b0	RO
6	New Register Request (NewRegReq) The application sets this bit for a new vendor control access.	1'b0	R_WS_SC
7:8	Reserved	2'b0	
9	Register Write (RegWr) Set this bit for register writes, and clear it for register reads.	1'b0	R_W
10:15	Register Address (RegAddr) The 6-bit PHY register address for immediate PHY Register Set access. Set to 6'h2F for Extended PHY Register Set access.	6'b0	R_W
16:23	UTMI+ Vendor Control Register Address (VCtrl) ULPI Extended Register Address (ExtRegAddr) The 6-bit PHY extended register address.	0x00	R_W
24:31	Register Data (RegData) Contains the write data for register write. Read data for register read, valid when VStatus Done is set.	0x00	R_W

38.5.2.13 ID Register (USB0_GSNPSID)

This is a read-only register that contains the release number of the function being used.

Figure 38-16. ID Register (USB0_GSNPSID)

Field	Description	Reset	Access
0:31	IP Core ID Release number of the function being used, currently OT2.60a.	0x4F54 <version>	RO

38.5.2.14 Host Periodic Transmit FIFO Size Register (USB0_HPTXFSIZ)

This register holds the size and the memory start address of the Periodic TxFIFO.

Figure 38-17. Host Periodic Transmit FIFO Size Register (USB0_HPTXFSIZ)

Field	Description	Reset	Access
0:15	Host Periodic TxFIFO Depth (PTxFSize) This value is in terms of 32-bit words. • Minimum value is 16 • Maximum value is 32,768 The power-on reset value of this register is specified as the Largest Host Mode Periodic Tx Data FIFO Depth = 512. You can write a new value in this field. Programmed values must not exceed the power-on value.	User selected	RO/R_W
16:31	Host Periodic TxFIFO Start Address (PTxFStAddr) The power-on reset value of this register is the sum of the Largest Rx Data FIFO Depth and Largest Non-periodic Tx Data FIFO Depth = 787. If you have programmed new values for the RxFIFO or Non-periodic TxFIFO, you can write their sum in this field. Programmed values must not exceed the power-on value.	User selected	RO/R_W

38.5.2.15 Device Periodic Transmit FIFO n Size Register (USB0_DPTXFSIZn)

FIFO_number: $1 \leq n \leq 15$

Offset: 104h + (FIFO_number – 1) * 04h

This register holds the memory start address of each periodic TxFIFO to be implemented in Device mode. Each periodic FIFO holds the data for one periodic IN Endpoint. This register is repeated for each periodic FIFO instantiated.

Figure 38-18. Device Periodic Transmit FIFO n Register (USB0_DPTXFSIZn)

Field	Description	Reset	Access
0:15	Device Periodic TxFIFO Size (DPTxFSize) This value is in terms of 32-bit words. • Minimum value is 4 • Maximum value is 768 The value of this register is the Largest Device Mode Periodic Tx Data FIFO Depth = 768.	User selected	RO
16:31	Device Periodic TxFIFO RAM Start Address (DPTxFStAddr) Holds the start address in the RAM for this periodic FIFO. The power-on reset value of this register is the sum of the Largest Rx Data FIFO Depth, Largest Non-periodic Tx Data FIFO Depth = 787. If you have programmed new values for the RxFIFO Non-periodic TxFIFO, or Device Periodic TxFIFOs, you can write their sum in this field. Programmed values must not exceed the power-on value.	User selected	RO/R_W

38.5.3 Host Mode Registers

These registers affect the operation of the function in the Host mode. Host mode registers must not be accessed in Device mode, as the results are undefined. Host Mode registers can be categorized as follows:

Host Global Registers
- Host Configuration Register (HCFG)
- Host Frame Interval Register (HFIR)
- Host Frame Number/Frame Time Remaining Register (HFNUM)
- Host Periodic Transmit FIFO/Queue Status Register (HPTXSTS)
- Host All Channels Interrupt Register (HAINT)
- Host All Channels Interrupt Mask Register (HAINTMSK)
- Host Port Control and Status Register (HPRT)

Host Port Control and Status Register
- Host Channel-n Characteristics Register (HCCHARn)

Host Channel-Specific Registers
- Host Channel-n Characteristics Register (HCCHARn)
- Host Channel-n Split Control Register (HCSPLTn)
- Host Channel-n Interrupt Register (HCINTn)
- Host Channel-n Interrupt Mask Register (HCINTMSKn)
- Host Channel-n Transfer Size Register (HCTSIZn)
- Host Channel-n DMA Address Register (HCDMAn)

38.5.3.1 Host Configuration Register (USB0_HCFG)

This register configures the function after power-on. Do not make changes to this register after initializing the Host.

Figure 38-19. Host Configuration Register (USB0_HCFG)

Field	Description	Reset	Access
0:28	Reserved	30'h0	
29	FS- and LS-Only Support (FSLSSupp) The application uses this bit to control the function's enumeration speed. Using this bit, the application can make the function enumerate as a FS Host, even if the connected Device supports HS traffic. Do not make changes to this field after initial programming. • 1'b0: HS/FS/LS, based on the maximum speed supported by the connected Device • 1'b1: FS/LS-only, even if the connected Device can support HS	1'b0	R_W
30:31	FS/LS PHY Clock Select (FSLSPclkSel) When the function is in FS Host mode • 2'b00: PHY clock is running at 30/60 MHz • 2'b01: PHY clock is running at 48 MHz • Others: Reserved When the function is in LS Host mode • 2'b00: PHY clock is running at 30/60 MHz. When the ULPI PHY Low Power mode is not selected, use 30/60 MHz. • All others: Reserved	2'b0	R_W

38.5.3.2 Host Frame Interval Register (USB0_HFIR)

This register stores the frame interval information for the current speed to which the USB function has enumerated.

Figure 38-20. Host Frame Interval Register (USB0_HFIR)

Field	Description	Reset	Access
0:15	Reserved	16'h0	
16:31	Frame Interval (FrInt) The value that the application programs to this field specifies the interval between two consecutive SOFs (FS) or micro-SOFs (HS) or Keep-Alive tokens (HS). This field contains the number of PHY clocks that constitute the required frame interval. The default value set in this field for a FS operation when the PHY clock frequency is 60 MHz. The application can write a value to this register only after the Port Enable bit of the Host Port Control and Status register (HPRT.PrtEnaPort) has been set. If no value is programmed, the function calculates the value based on the PHY clock specified in the FS/LS PHY Clock Select field of the Host Configuration register (HCFG.FSLSPclkSel). Do not change the value of this field after the initial configuration. • 125µ (PHY clock frequency for HS) • 1ms PHY clock frequency for FS/LS)	16'd60000	R_W

38.5.3.3 Host Frame Number/Frame Time Remaining Register (USB0_HFNUM)

This register indicates the current frame number. It also indicates the time remaining (in terms of the number of PHY clocks) in the current (micro)frame.

Figure 38-21. Host Frame Number/Frame Time Remaining Register (USB0_HFNUM)

Field	Description	Reset	Access
0:15	Frame Time Remaining (FrRem) Indicates the amount of time remaining in the current microframe (HS) or frame (FS/LS), in terms of PHY clocks. This field decrements on each PHY clock. When it reaches zero, this field is reloaded with the value in the Frame Interval register and a new SOF is transmitted on the USB.	16'h0	RW
16:31	Frame Number (FrNum) This field increments when a new SOF is transmitted on the USB, and is reset to 0 when it reaches 16'h3FFF. Reads return the frame number value.	16'h3FFF	RO

38.5.3.4 Host Periodic Transmit FIFO/Queue Status Register (USB0_HPTXSTS)

This read-only register contains the free space information for the Periodic TxFIFO and the Periodic Transmit Request Queue.

.

Figure 38-22. Host Periodic Transmit FIFO/Queue Status Register (USB0_HPTXSTS)

Field	Description	Reset	Access
0:7	Top of the Periodic Transmit Request Queue (PTxQTop) This indicates the entry in the Periodic Tx Request Queue that is currently being processes by the MAC. This register is used for debugging. • Bit 0: Odd/Even (micro)frame - 1'b0: send in even (micro)frame - 1'b1: send in odd (micro)frame • Bits 1:4: Channel/Endpoint number • Bits 5:6: Type - 2'b00: IN/OUT - 2'b01: Zero-length packet - 2'b10: CSPLIT - 2'b11: Disable channel command • Bit 7: Terminate (last entry for the selected channel/Endpoint)	8'h0	RO
8:15	Periodic Transmit Request Queue Space Available (PTxQSpcAvail) Indicates the number of free locations available to be written in the Periodic Transmit Request Queue. This queue holds both IN and OUT requests. • 8'h0: Periodic Transmit Request Queue is full • 8'h1: 1 location available • 8'h2: 2 locations available • n: n locations available ($0 \le n \le 8$) • Others: Reserved	User selected RO	RO
16:31	Periodic Transmit Data FIFO Space Available (PTxFSpcAvail) Indicates the number of free locations available to be written to in the Periodic TxFIFO. Values are in terms of 32-bit words • 16'h0: Periodic TxFIFO is full • 16'h1: 1 word available • 16'h2: 2 words available • 16'h n: n words available (where $0 \le n \le 32,768$) • 16'h8000: 32,768 words available • Others: Reserved	User selected RW	RW

38.5.3.5 Host All Channels Interrupt Register (USB0_HAINT)

When a significant event occurs on a channel, the Host All Channels Interrupt register interrupts the application using the Host Channels Interrupt bit of the Interrupt register (GINTSTS.HChInt). There is one interrupt bit per channel, up to a maximum of 16 bits. Bits in this register are set and cleared when the application sets and clears bits in the corresponding Host Channel-n Interrupt register.

Figure 38-23. Host All Channels Interrupt Register (USB0_HAINT)

Field	Description	Reset	Access
0:15	Reserved	16'h0	
16:31	Channel Interrupts (HAINT) One bit per channel: Bit 31 for Channel 0, bit 16 for Channel 15	16'h0	RO

38.5.3.6 Host All Channels Interrupt Mask Register (USB0_HAINTMSK)

The Host All Channel Interrupt Mask register works with the Host All Channel Interrupt register to interrupt the application when an event occurs on a channel. There is one interrupt mask bit per channel, up to a maximum of 16 bits.

- Mask interrupt: 1'b0
- Unmask interrupt: 1'b1

Figure 38-24. Host All Channels Interrupt Mask Register (USB0_HAINTMSK)

Field	Description	Reset	Access
0:15	Reserved	16'h0	
16:31	Channel Interrupt Mask (HAINTMsk) One bit per channel: Bit 31 for channel 0, bit 16 for channel 15	16'h0	R_W

38.5.3.7 Host Port Control and Status Register (USB0_HPRT)

This register is available only in Host mode. Currently, the OTG Host supports only one port.

A single register holds USB port-related information such as USB reset, enable, suspend, resume, connect status, and test mode for each port. The R_SS_WC bits in this register can trigger an interrupt to the application through the Host Port Interrupt bit of the Interrupt register (GINTSTS.PrtInt). On a Port Interrupt, the application must read this register and clear the bit that caused the interrupt. For the R_SS_WC bits, the application must write a 1 to the bit to clear the interrupt.

Figure 38-25. Host Port Control and Status Register (USB0_HPRT)

Field	Description	Reset	Access
0:12	Reserved	12'h0	
13:14	Port Speed (PrtSpd) Indicates the speed of the Device attached to this port. • 2'b00: High speed • 2'b01: Full speed • 2'b10: Low speed • 2'b11: Reserved	2'b0	RO

Figure 38-25. Host Port Control and Status Register (USB0_HPRT) (Continued)

Field	Description	Reset	Access
15:18	Port Test Control (PrtTstCtl) The application writes a non-zero value to this field to put the port into a Test mode, and the corresponding pattern is signaled on the port. • 4'b0000: Test mode disabled • 4'b0001: Test_J mode • 4'b0010: Test_K mode • 4'b0011: Test_SE0_NAK mode • 4'b0100: Test_Packet mode • 4'b0101: Test_Force_Enable • Others: Reserved	4'h0	R_W
19	Port Power (PrtPwr) The application uses this field to control power to this port, and the function clears this bit on an over current condition. • 1'b0: Power off • 1'b1: Power on	1'b0	R_W_SC
20:21	Port Line Status (PrtLnSts) Indicates the current logic level USB data lines • Bit 20: Logic level of D + • Bit 21: Logic level of D–	2'b0	RO
22	Reserved	1'b0	
23	Port Reset (PrtRst) When the application sets this bit, a reset sequence is started on this port. The application must time the reset period and clear this bit after the reset sequence is complete. • 1'b0: Port not in reset • 1'b1: Port in reset The application must leave this bit set for at least a minimum duration mentioned below to start a reset on the port. The application can leave it set for another 10 ms in addition to the required minimum duration, before clearing the bit, even though there is no maximum limit set by the USB standard. • High speed: 50 ms • Full speed/Low speed: 10 ms	1'b0	R_W
24	Port Suspend (PrtSusp) The application sets this bit to put this port in Suspend mode. The function only stops sending SOFs when this is set. To stop the PHY clock, the application must set the Port Clock Stop bit, which asserts the suspend input pin of the PHY. The read value of this bit reflects the current suspend status of the port. This bit is cleared by the function after a remote wake up signal is detected or the application sets the Port Reset bit or Port Resume bit in this register or the Resume/Remote Wake up Detected Interrupt bit or Disconnect Detected Interrupt bit in the Interrupt register (GINTSTS.WkUpInt or GINTSTS.DisconnInt, respectively). • 1'b0: Port not in Suspend mode • 1'b1: Port in Suspend mode	1'b0	R_WS_SC
25	Port Resume (PrtRes) The application sets this bit to drive resume signaling on the port. The function continues to drive the resume signal until the application clears this bit. If the function detects a USB remote wake up sequence, as indicated by the Port Resume/Remote Wake up Detected Interrupt bit of the Interrupt register (GINTSTS.WkUpInt), the function starts driving resume signaling without application intervention and clears this bit when it detects a disconnect condition. The read value of this bit indicates whether the function is currently driving resume signaling. • 1'b0: No resume driven	1'b0	R_W_SS_SC

Figure 38-25. Host Port Control and Status Register (USB0_HPRT) (Continued)

Field	Description	Reset	Access
26	Port Over current Change (PrtOvrCurrChng) The function sets this bit when the status of the Port Over Current Active bit in this register changes.	1'b0	R_SS_WC
27	Port Over Current Active (PrtOvrCurrAct) Indicates the over current condition of the port. • 1'b0: No over current condition • 1'b1: Over current condition	1'b0	RO
28	Port Enable/Disable Change (PrtEnChng) The function sets this bit when the status of the Port Enable bit of this register changes.	1'b0	R_SS_WC
29	Port Enable (PrtEna) A port is enabled only by the function after a reset sequence, and is disabled by an over current condition, a disconnect condition, or by the application clearing this bit. The application cannot set this bit by a register write. It can only clear it to disable the port. This bit does not trigger any interrupt to the application. • 1'b0: Port disabled • 1'b1: Port enabled	1'b0	R_SS_SC_WC
30	Port Connect Detected (PrtConnDet) The function sets this bit when a Device connection is detected to trigger an interrupt to the application using the Host Port Interrupt bit of the Interrupt register (GINTSTS.PrtInt). The application must write a 1 to this bit to clear the interrupt.	1'b0	R_SS_WC
31	Port Connect Status (PrtConnSts) • 0: No Device is attached to the port. • 1: A Device is attached to the port.	1'b0	RO

38.5.3.8 Host Channel n Characteristics Register (USB0_HCCHARn)

Channel_number: $0 \leq n \leq 2$

Offset: 500h + (Channel_number * 20h)

Figure 38-26. Host Channel n Characteristics Register (USB0_HCCHARn)

Field	Description	Reset	Access
0	Channel Enable (ChEna) This field is set by the application and cleared by the OTG Host. 1'b0: Channel disabled 1'b1: Channel enabled	1'b0	R_WS_SC
1	Channel Disable (ChDis) The application sets this bit to stop transmitting/receiving data on a channel, even before the transfer for that channel is complete. The application must wait for the Channel Disabled interrupt before treating the channel as disabled.	1'b0	R_WS_SC
2	Odd Frame (OddFrm) This field is set (reset) by the application to indicate that the OTG Host must perform a transfer in an odd (micro)frame. This field is applicable for only periodic (isochronous and interrupt) transactions. 1'b0: Even (micro)frame 1'b1: Odd (micro)frame	1'b0	R_W
3:9	Device Address (DevAddr) This field selects the specific Device serving as the data source or sink.	7'h0	R_W

Figure 38-26. Host Channel n Characteristics Register (USB0_HCCHARn) (Continued)

Field	Description	Reset	Access
10:11	Multi Count (MC)/Error Count (EC) When the Split Enable bit of the Host Channel-n Split Control register (HCSPLTn.SpltEna) is reset (1'b0), this field indicates to the Host the number of transactions that must be executed per microframe for this Endpoint. • 2'b00: Reserved This field yields undefined results. • 2'b01: 1 transaction • 2'b10: 2 transactions to be issued for this Endpoint per microframe • 2'b11: 3 transactions to be issued for this Endpoint per microframe When HCSPLTn.SpltEna is set (1'b1), this field indicates the number of immediate retries to be performed for a periodic split transactions on transaction errors. This field must be set to at least 2'b01.	2'b0	R_W
12:13	Endpoint Type (EPType) Indicates the transfer type selected. • 2'b00: Control • 2'b01: Isochronous • 2'b10: Bulk • 2'b11: Interrupt	2'b0	R_W
14	Low-Speed Device (LSpdDev) This field is set by the application to indicate that this channel is communicating to a low-speed Device.	1'b0	R_W
15	Reserved	1'b0	
16	Endpoint Direction (EPDir) Indicates whether the transaction is IN or OUT. • 1'b0: OUT • 1'b1: IN	1'b0	R_W
17:20	Endpoint Number (EPNum) Indicates the Endpoint number on the Device serving as the data source or sink.	4'h0	R_W
21:31	Maximum Packet Size (MPS) Indicates the maximum packet size of the associated Endpoint.	11'h0	R_W

38.5.3.9 Host Channel n Split Control Register (USB0_HCSPLTn)

Channel_number: $0 \leq n \leq 2$

Offset: 504h + (Channel_number * 20h)

Figure 38-27. Host Channel n Split Control Register (USB0_HCSPLTn)

Field	Description	Reset	Access
0	Split Enable (SpltEna) The application sets this field to indicate that this channel is enabled to perform split transactions.	1'b0	R_W
1:14	Reserved	14'h0	
15	Do Complete Split (CompSplt) The application sets this field to request the OTG Host to perform a complete split transaction.	1'b0	R_W

Figure 38-27. Host Channel n Split Control Register (USB0_HCSPLTn) (Continued)

Field	Description	Reset	Access
16:17	Transaction Position (XactPos) This field is used to determine whether to send all, first, middle, or last payloads with each OUT transaction. • 2'b11: All. This is the entire data payload is of this transaction (which is less than or equal to 188B). • 2'b10: Begin. This is the first data payload of this transaction (which is larger than 188B). • 2'b00: Mid. This is the middle payload of this transaction (which is larger than 188B). • 2'b01: End. This is the last payload of this transaction (which is larger than 188B).	2'h0	R_W
18:24	Hub Address (HubAddr) This field holds the Device address of the transaction translator's hub.	7'h0	R_W
25:31	Port Address (PrtAddr) This field is the port number of the recipient transaction translator.	7'h0	R_W

38.5.3.10 Host Channel n Interrupt Register (USB0_HCINTn)

Channel_number: $0 \leq n \leq 2$

Offset: 508h + (Channel_number * 20h)

This register indicates the status of a channel with respect to USB- and AHB-related events. The application must read this register when the Host Channels Interrupt bit of the Interrupt register (GINTSTS.HChInt) is set. Before the application can read this register, it must first read the Host All Channels Interrupt (HAINT) register to get the exact channel number for the Host Channel-n Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the HAINT and GINTSTS registers.

Figure 38-28. Host Channel n Interrupt Register (USB0_HCINTn)

Field	Description	Reset	Access
0:20	Reserved		
21	Data Toggle Error (DataTglErr)	1'b0	R_SS_WC
22	Frame Overrun (FrmOvrun)	1'b0	R_SS_WC
23	Babble Error (BblErr)	1'b0	R_SS_WC
24	Transaction Error (XactErr) Indicates one of the following errors occurred on the USB. • CRC check failure • Timeout • Bit stuff error • False EOP	1'b0	R_SS_WC
25	NYET Response Received Interrupt (NYET)	1'b0	R_SS_WC
26	ACK Response Received Interrupt (ACK)	1'b0	R_SS_WC
27	NAK Response Received Interrupt (NAK)	1'b0	R_SS_WC
28	STALL Response Received Interrupt (STALL)	1'b0	R_SS_WC

Figure 38-28. Host Channel n Interrupt Register (USB0_HCINTn) (Continued)

Field	Description	Reset	Access
29	AHB Error (AHBErr) This is generated only in Internal DMA mode when there is an AHB error during AHB read/write. The application can read the corresponding channel's DMA address register to get the error address.	1'b0	R_SS_WC
30	Channel Halted (ChHltd) Indicates the transfer completed abnormally either because of any USB transaction error or in response to disable request by the application.	1'b0	R_SS_WC
31	Transfer Completed (XferCompl) Transfer completed normally without any errors.	1'b0	R_SS_WC

38.5.3.11 Host Channel n Interrupt Mask Register (USB0_HCINTMSKn)

Channel_number: $0 \leq n \leq 2$

Offset: 50Ch + (Channel_number * 20h)

This register reflects the mask for each channel status described in the previous section.

- Mask interrupt: 1'b0
- Unmask interrupt: 1'b1

Figure 38-29. Host Channel n Interrupt Mask Register (USB0_HCINTMSKn)

Field	Description	Reset	Access
0:20	Reserved	21'h0	
21	Data Toggle Error Mask (DataTglErrMsk)	1'b0	R_W
22	Frame Overrun Mask (FrmOvrunMsk)	1'b0	R_W
23	Babble Error Mask (BblErrMsk)	1'b0	R_W
24	Transaction Error Mask (XactErrMsk)	1'b0	R_W
25	NYET Response Received Interrupt Mask (NyetMsk)	1'b0	R_W
26	ACK Response Received Interrupt Mask (AckMsk)	1'b0	R_W
27	NAK Response Received Interrupt Mask (NakMsk)	1'b0	R_W
28	STALL Response Received Interrupt Mask (StallMsk)	1'b0	R_W
29	AHB Error Mask (AHBErrMsk)	1'b0	R_W
30	Channel Halted Mask (ChHltdMsk)	1'b0	R_W
31	Transfer Completed Mask (XferComplMsk)	1'b0	R_W

38.5.3.12 Host Channel n Transfer Size Register (USB0_HCTSIZn)

Channel_number: 0 ≤ n ≤ 2

Offset: 510h + (Channel_number * 20h)

Figure 38-30. Host Channel n Transfer Size Register (USB0_HCTSIZn)

Field	Description	Reset	Access
0	Do Ping (DoPng) Setting this field to 1 directs the Host to do PING protocol.	1'h0	R_W
1:2	PID (Pid) 2'b0 R_W The application programs this field with the type of PID to use for the initial transaction. The Host maintains this field for the rest of the transfer. • 2'b00: DATA0 • 2'b01: DATA2 • 2'b10: DATA1 • 2'b11: MDATA (non-control)/SETUP (control)		
3:12	Packet Count (PktCnt) This field is programmed by the application with the expected number of packets to be transmitted (OUT) or received (IN). The Host decrements this count on every successful transmission or reception of an OUT/IN packet. Once this count reaches zero, the application is interrupted to indicate normal completion. The width of this counter is 10.	10'b0	R_W
13:31	Transfer Size (XferSize) For an OUT, this field is the number of data bytes the Host sends during the transfer. For an IN, this field is the buffer size that the application has Reserved for the transfer. The application is expected to program this field as an integer multiple of the maximum packet size for IN transactions (periodic and non-periodic). The width of this counter is 19.	19'b0	R_W

38.5.3.13 Host Channel n DMA Address Register (USB0_HCDMAn)

Channel_number: 0 ≤ n ≤ 3

Offset: 514h + (Channel_number * 20h)

This register is used by the OTG Host in the internal DMA mode to maintain the current buffer pointer for IN/OUT transactions. The starting DMA address must be DWORD-aligned.

Figure 38-31. Host Channel n DMA Address Register (USB0_HCDMAn)

Field	Description	Reset	Access
0:31	DMA Address (DMAAddr) This field holds the start address in the external memory from which the data for the Endpoint must be fetched or to which it must be stored. This register is incremented on every AHB transaction.	32'h0	R_W

38.5.4 Device Mode Registers

These registers are visible only in Device mode and must not be accessed in Host mode, as the results are unknown. Some of them affect all the endpoints uniformly, while others affect only a specific Endpoint. Device Mode registers fall into two categories:

Device Global Registers
- Device Configuration Register (DCFG)
- Device Control Register (DCTL)
- Device Status Register (DSTS)
- Device IN Endpoint Common Interrupt Mask Register (DIEPMSK)
- Device OUT Endpoint Common Interrupt Mask Register (DOEPMSK)
- Device All Endpoints Interrupt Register (DAINT)
- Device All Endpoints Interrupt Mask Register (DAINTMSK)
- Device IN Token Sequence Learning Queue Read Register 1 (DTKNQR1)
- Device IN Token Sequence Learning Queue Read Register 2 (DTKNQR2)
- Device IN Token Sequence Learning Queue Read Register 3 (DTKNQR3)
- Device IN Token Sequence Learning Queue Read Register 4 (DTKNQR4)
- Device VBUS Discharge Time Register (DVBUSDIS)
- Device VBUS Pulsing Time Register (DVBUSPULSE)
- Device IN Endpoint FIFO Empty Interrupt Mask Register (DIEPEMPMSK)
- Device Control IN Endpoint 0 Control Register (DIEPCTL0)

Device Logical Endpoint-Specific Registers
- Device Control OUT Endpoint 0 Control Register (DOEPCTL0)
- Device Endpoint-n Control Register (DIEPCTLn/DOEPCTLn)
- Device Endpoint-n Interrupt Register (DIEPINTn/DOEPINTn)
- Device Endpoint 0 Transfer Size Register (DIEPTSIZ0/DOEPTSIZ0)
- Device Endpoint-n Transfer Size Register (DIEPTSIZn/DOEPTSIZn)
- Device Endpoint-n DMA Address Registers (DIEPDMAn/DOEPDMAn)
- Device IN Endpoint Transmit FIFO Status Register (DTXFSTSn)

38.5.4.1 Device Configuration Register (USB0_DCFG)

This register configures the function in Device mode after power-on or after certain control commands or enumeration. Do not make changes to this register after initial programming.

Figure 38-32. Device Configuration Register (USB0_DCFG)

Field	Description	Reset	Access
0:8	Reserved	9'h0	
9:13	IN Endpoint Mismatch Count (EPMisCnt) This field is valid only in shared FIFO operation. The application programs this filed with a count that determines when the function generates an Endpoint Mismatch interrupt (GINTSTS.EPMis). The function loads this value into an internal counter and decrements it. The counter is reloaded whenever there is a match or when the counter expires. The width of this counter depends on the depth of the Token Queue.	5'h8	R_W
14:18	Reserved 5'h0		

Figure 38-32. Device Configuration Register (USB0_DCFG) (Continued)

Field	Description	Reset	Access
19:20	Periodic Frame Interval (PerFrInt) Indicates the time within a (micro)frame at which the application must be notified using the End Of Periodic Frame Interrupt. This can be used to determine if all the isochronous traffic for that (micro)frame is complete. 00 80% of the (micro)frame interval 01 85% 10 90% 11 95%	2'h0	R_W
21:27	Device Address (DevAddr) The application must program this field after every SetAddress control command.	7'h0	R_W
28	Reserved	1'b0	
29	Non-Zero-Length Status OUT Handshake (NZStsOUTHShk) The application can use this field to select the handshake the function sends on receiving a non-zero-length data packet during the OUT transaction of a control transfer's Status stage. 0 Send the received OUT packet to the application (zero length or nonzero length) and send a handshake based on the NAK and STALL bits for the Endpoint in the Device Endpoint Control register. 1 Send a STALL handshake on a non-zero length status OUT transaction and do not send the received OUT packet to the application.	1'b0	R_W
30:31	Device Speed (DevSpd) Indicates the speed at which the application requires the function to enumerate, or the maximum speed the application can support. However, the actual bus speed is determined only after the chirp sequence is completed, and is based on the speed of the USB Host to which the function is connected. 00 High speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) 01 Full speed (USB 2.0 PHY clock is 30 MHz or 60 MHz) 10 Reserved 11 Full speed (USB 1.1 transceiver clock is 48 MHz)	2'b0	R_W

38.5.4.2 Device Control Register (USB0_DCTL)

Figure 38-33. Device Control Register (USB0_DCTL)

Field	Description	Reset	Access
0:20	Reserved	0	
21	Clear Global OUT NAK (CGOUTNak) A write to this field clears the Global OUT NAK.	1'b0	WO
22	Set Global OUT NAK (SGOUTNak) A write to this field sets the Global OUT NAK. The application uses this bit to send a NAK handshake on all OUT endpoints. The application must set the this bit only after making sure that the Global OUT NAK Effective bit in the Interrupt Register (GINTSTS.GOUTNakEff) is cleared.	1'b0	WO
23	Clear Global Non-periodic IN NAK (CGNPInNak) A write to this field clears the Global Non-periodic IN NAK.	1'b0	WO

Figure 38-33. Device Control Register (USB0_DCTL) (Continued)

Field	Description	Reset	Access
24	Set Global Non-periodic IN NAK (SGNPInNak) A write to this field sets the Global Non-periodic IN NAK. The application uses this bit to send a NAK handshake on all non-periodic IN endpoints. The function can also set this bit when a timeout condition is detected on a non-periodic Endpoint in shared FIFO operation. The application must set this bit only after making sure that the Global IN NAK Effective bit in the Interrupt Register (GINTSTS.GINNakEff) is cleared.	1'b0	WO
25:27	Test Control (TstCtl) • 3'b000: Test mode disabled • 3'b001: Test_J mode • 3'b010: Test_K mode • 3'b011: Test_SE0_NAK mode • 3'b100: Test_Packet mode • 3'b101: Test_Force_Enable • Others: Reserved	3'b0	R_W
28	Global OUT NAK Status (GOUTNakSts) • 1'b0: A handshake is sent based on the FIFO Status and the NAK and STALL bit settings. • 1'b1: No data is written to the RxFIFO, irrespective of space availability. Sends a NAK handshake on all packets, except on SETUP transactions. All isochronous OUT packets are dropped.	1'b0	RO
29	Global Non-periodic IN NAK Status (GNPINNakSts) • 1b0: A hand shake is sent out based on the data availability in the transmit FIFO. • 1b1: A NAK handshake is sent out on all non-periodic IN endpoints, irrespective of the data availability in the transmit FIFO.	1'b0	RO
30	Soft Disconnect (SftDiscon) The application uses this bit to signal the USB function to do a soft disconnect. As long as this bit is set, the Host does not see that the Device is connected, and the Device does not receive signals on the USB. The function stays in the disconnected state until the application clears this bit. The minimum duration for which the function must keep this bit set is specified in *Table 38-7*: • 1'b0: Normal operation. When this bit is cleared after a soft disconnect, the function drives the phy_opmode_o signal on the UTMI+ to 2'b00, which generates a Device connect event to the USB Host. When the Device is reconnected, the USB Host restarts Device enumeration. • 1'b1: The function drives the phy_opmode_o signal on the UTMI+ to 2'b01, which generates a Device disconnect event to the USB Host.	1'b0	R_W
31	Remote Wake Up Signaling (RmtWkUpSig) When the application sets this bit, the function initiates remote signaling to wake up the USB Host. The application must set this bit to instruct the function to exit the Suspend state. As specified in the USB 2.0 specification, the application must clear this bit 1–15 ms after setting it.	1'b0	R_W

The following table lists the minimum duration under various conditions for which the SoftDisconnect (SftDiscon) bit must be set for the USB Host to detect a Device disconnect. To accommodate clock jitter, it is recommended that the application add some extra delay to the specified minimum duration.

Table 38-7. Minimum Duration for Soft Disconnect

Operating Speed	Device State	Minimum Duration
High speed	Suspended	1 ms + 2.5 μs
High speed	Idle	3 ms + 2.5 μs
High speed	Not Idle or Suspended (performing transactions)	12.5 μs
Full speed/Low speed	Suspended	1 ms + 2.5 μs
Full speed/Low speed	Idle	2.5 μs
Full speed/Low speed	Not Idle or Suspended (performing transactions)	2.5 μs

38.5.4.3 Device Status Register (USB0_DSTS)

This register indicates the status of the function with respect to USB-related events. It must be read on interrupts from Device All Interrupts (DAINT) register.

Figure 38-34. Device Status Register (USB0_DSTS)

Field	Description	Reset	Access
0:9	Reserved	10'h0	
10:23	Frame or Microframe Number of the Received SOF (SOFFN) When the function is operating at high speed, this field contains a microframe number. When the function is operating at full or low speed, this field contains a frame number.	14'h0	RO
24:27	Reserved	4'h0	
28	Erratic Error (ErrticErr) The function sets this bit to report any erratic errors (phy_rxvalid_i/phy_rxvldh_i or phy_rxactive_i is asserted for at least 2 ms, due to PHY error) seen on the UTMI+. Due to erratic errors, the USB function goes into Suspended state and an interrupt is generated to the application with Early Suspend bit of the Interrupt register (GINTSTS.ErlySusp). If the early suspend is asserted due to an erratic error, the application can only perform a soft disconnect recover.	1'b0	RO
29:30	Enumerated Speed (EnumSpd) Indicates the speed at which the USB function has come up after speed detection through a chirp sequence. • 2'b00: High speed (PHY clock is running at 30 or 60 MHz) • 2'b01: Full speed (PHY clock is running at 30 or 60 MHz) • 2'b10: Low speed (PHY clock is running at 6 MHz) • 2'b11: Full speed (PHY clock is running at 48 MHz)	2'h0	RO
31	Suspend Status (SuspSts) In Device mode, this bit is set as long as a Suspend condition is detected on the USB. The function enters the Suspended state when there is no activity on the phy_line_state_i signal for an extended period of time. The function comes out of the suspend: • When there is any activity on the phy_line_state_i signal • When the application writes to the Remote Wakeup Signaling bit in the Device Control register (DCTL.RmtWkUpSig).	1'b0	RO

38.5.4.4 Device IN Endpoint Common Interrupt Mask Register (USB0_DIEPMSK)

This register works with each of the Device IN Endpoint Interrupt (DIEPINTn) registers for all endpoints to generate an interrupt per IN Endpoint. The IN Endpoint interrupt for a specific status in the DIEPINTn register can be masked by writing to the corresponding bit in this register. Status bits are masked by default.

- Mask interrupt: 1'b0
- Unmask interrupt: 1'b1

Figure 38-35. Device IN Endpoint Common Interrupt Mask Register (USB0_DIEPMSK)

Field	Description	Reset	Access
0:22	Reserved	23'h0	
23	Fifo Under run Mask (TxfifoUndrnMsk)	1'b0	R_W
24	Reserved.	1'b0	R_W
25	IN Endpoint NAK Effective Mask (INEPNakEffMsk)	1'b0	R_W
26	IN Token received with EP Mismatch Mask (INTknEPMisMsk)	1'b0	R_W
27	IN Token Received When TxFIFO Empty Mask (INTknTXFEmpMsk)	1'b0	R_W
28	Timeout Condition Mask (TimeOUTMsk) (Non-isochronous endpoints)	1'b0	R_W
29	AHB Error Mask (AHBErrMsk)	1'b0	R_W
30	Endpoint Disabled Interrupt Mask (EPDisbldMsk)	1'b0	R_W
31	Transfer Completed Interrupt Mask (XferComplMsk)	1'b0	R_W

38.5.4.5 Device OUT Endpoint Common Interrupt Mask Register (USB0_DOEPMSK)

This register works with each of the Device OUT Endpoint Interrupt (DOEPINTn) registers for all endpoints to generate an interrupt per OUT Endpoint. The OUT Endpoint interrupt for a specific status in the DOEPINTn register can be masked by writing into the corresponding bit in this register. Status bits are masked by default.

- Mask interrupt: 1'b0
- Unmask interrupt: 1'b1

Figure 38-36. Device OUT Endpoint Common Interrupt Mask Register (USB0_DOEPMSK)

Field	Description	Reset	Access
0:22	Reserved	23'h0	
23	OUT Packet Error Mask (OutPktErrMsk)	1'b0	R_W
24	Reserved	1'b0	
25	Back-to-Back SETUP Packets Received Mask (Back2BackSETup) Applies to control OUT endpoints only.	1'b0	R_W

Figure 38-36. Device OUT Endpoint Common Interrupt Mask Register (USB0_DOEPMSK) (Continued)

Field	Description	Reset	Access
26	Reserved	1'b0	
27	OUT Token Received when Endpoint Disabled Mask (OUTTknEPdisMsk) Applies to control OUT endpoints only.	1'b0	R_W
28	SETUP Phase Done Mask (SetUPMsk) Applies to control endpoints only.	1'b0	R_W
29	AHB Error (AHBErrMsk)	1'b0	R_W
30	Endpoint Disabled Interrupt Mask (EPDisbldMsk)	1'b0	R_W
31	Transfer Completed Interrupt Mask (XferComplMsk)	1'b0	R_W

38.5.4.6 Device All Endpoints Interrupt Register (USB0_DAINT)

When a significant event occurs on an Endpoint, a Device All Endpoints Interrupt register interrupts the application using the Device OUT Endpoints Interrupt bit or Device IN Endpoints Interrupt bit of the Interrupt register (GINTSTS.OEPInt or GINTSTS.IEPInt, respectively). There is one interrupt bit per Endpoint, up to a maximum of 16 bits for OUT endpoints and 16 bits for IN endpoints. For a bidirectional Endpoint, the corresponding IN and OUT interrupt bits are used. Bits in this register are set and cleared when the application sets and clears bits in the corresponding Device Endpoint-n Interrupt register (DIEPINTn/DOEPINTn).

Figure 38-37. Device All Endpoints Interrupt Register (USB0_DAINT)

Field	Description	Reset	Access
0:12	Reserved	13'h0	RO
13:15	OUT Endpoint Interrupt Bits (OutEPInt) One bit per OUT Endpoint: Bit 15 for OUT Endpoint 0, bit 13 for OUT Endpoint 2	3'h0	RO
16:28	Reserved	13'h0	RO
29:31	IN Endpoint Interrupt Bits (InEpInt) One bit per IN Endpoint: Bit 31 for IN Endpoint 0, bit 29 for Endpoint 2	3'h0	RO

38.5.4.7 Device All Endpoints Interrupt Mask Register (USB0_DAINTMSK)

The Device Endpoint Interrupt Mask register works with the Device Endpoint Interrupt register to interrupt the application when an event occurs on a Device Endpoint. However, the Device All Endpoints Interrupt (DAINT) register bit corresponding to that interrupt is still set.

- Mask Interrupt: 1'b0
- Unmask Interrupt: 1'b1

Figure 38-38. Device Endpoints Interrupt Mask Register (USB0_DAINTMSK)

Field	Description	Reset	Access
0:12	Reserved	13'h0	RO
13:15	OUT EP Interrupt Mask Bits (OutEpMsk) One per OUT Endpoint: Bit 15 for OUT EP 0, bit 13 for OUT EP 2	3'h0	R_W
16:28	Reserved	13'h0	RO
29:31	IN EP Interrupt Mask Bits (InEpMsk One bit per IN Endpoint: Bit 31 for IN EP 0, bit 29 for IN EP 2	3'h0	R_W

38.5.4.8 Device IN Token Sequence Learning Queue Read Register 1 (USB0_DTKNQR1)

The depth of the IN Token Sequence Learning Queue is eight. The queue is four bits wide to store the Endpoint number. A read from this register returns the first five Endpoint entries of the IN Token Sequence Learning Queue. When the queue is full, the new token is pushed into the queue and oldest token is discarded.

Figure 38-39. Device IN Token Sequence Learning Queue Read Register 1 (USB0_DTKNQR1)

Field	Description	Reset	Access
0:23	Endpoint Token (EPTkn) Four bits per token represent the Endpoint number of the token: • Bits 0:3: Endpoint number of Token 5 • Bits 4:7: Endpoint number of Token 4 ..... • Bits 16:19: Endpoint number of Token 1 • Bits 20:23: Endpoint number of Token 0	24'h0	RO
24	Wrap Bit (WrapBit) This bit is set when the write pointer wraps. It is cleared when the learning queue is cleared.	1'b0	RO
25:26	Reserved	2'h0	RO
27:31	IN Token Queue Write Pointer (INTknWPtr)	5'h0	RO

38.5.4.9 Device IN Token Sequence Learning Queue Read Register 2 (USB0_DTKNQR2)

A read from this register returns the next eight Endpoint entries of the learning queue.

Figure 38-40. Device IN Token Sequence Learning Queue Register 2 (USB0_DTKNQR2)

Field	Description	Reset	Access
0:31	Endpoint Token (EPTkn) Four bits per token represent the Endpoint number of the token: • Bits 0:3: Endpoint number of Token 13 • Bits 4:7: Endpoint number of Token 12 ..... • Bits 24:27: Endpoint number of Token 7 • Bits 28:31: Endpoint number of Token 6	32'h0	RO

38.5.4.10 Device IN Token Sequence Learning Queue Read Register 3 (USB0_DTKNQR3)

A read from this register returns the next eight Endpoint entries of the learning queue.

Figure 38-41. Device IN Token Sequence Learning Queue Register 3 (USB0_DTKNQR3)

Field	Description	Reset	Access
0:31	Endpoint Token (EPTkn) Four bits per token represent the Endpoint number of the token: • Bits 0:3: Endpoint number of Token 21 • Bits 4:7: Endpoint number of Token 20 ..... • Bits 24:27: Endpoint number of Token 15 • Bits 28:31: Endpoint number of Token 14	32'h0	RO

38.5.4.11 Device IN Token Sequence Learning Queue Read Register 4 (USB0_DTKNQR4)

A read from this register returns the last eight Endpoint entries of the learning queue.

Figure 38-42. Device IN Token Sequence Learning Queue Register 4 (USB0_DTKNQR4)

Field	Description	Reset	Access
0:31	Endpoint Token (EPTkn) Four bits per token represent the Endpoint number of the token: • Bits 0:3: Endpoint number of Token 29 • Bits 4:7: Endpoint number of Token 28 ..... • Bits 24:27: Endpoint number of Token 23 • Bits 28:31: Endpoint number of Token 22	32'h0	RO

38.5.4.12 Device VBUS Discharge Time Register (USB0_DVBUSDIS)

This register specifies the VBUS discharge time after V BUS pulsing during SRP.

Figure 38-43. Device VBUS Discharge Time Register (USB0_DVBUSDIS)

Field	Description	Reset	Access
0:15	Reserved	16'h0	
16:31	Device V_{BUS} Discharge Time (DVBUSDis) Specifies the V_{BUS} discharge time after V_{BUS} pulsing during SRP. This value equals V_{BUS} discharge time in PHY clocks/1024. The value you use depends on the PHY operating at 60MHz (8-bit data width). Depending on your V BUS load, this value can need adjustment.	60 MHz: 16'h17D7	R_W

38.5.4.13 Device VBUS Pulsing Time Register (USB0_DVBUSPULSE)

This register specifies the VBUS pulsing time during SRP.

Figure 38-44. Device VBUS Pulsing Time Register (USB0_DVBUSPULSE)

Field	Description	Reset	Access
0:19	Reserved	20'h0	
20:31	Device V_{BUS} Pulsing Time (DVBUSPulse) Specifies the V_{BUS} pulsing time during SRP. This value equals V_{BUS} pulsing time in PHY clocks/1024. The value you use depends on the PHY operating at 60MHz (8-bit data width).	60 MHz: 12'h5B8	R_W

There is one set of Endpoint registers per logical Endpoint. A logical Endpoint is unidirectional: it can be either IN or OUT. To represent a bidirectional Endpoint, two logical endpoints are required, one for the IN direction and the other for the OUT direction. This is also true for control endpoints. The registers and register fields described in this section can pertain to IN or OUT endpoints, or both, or specific Endpoint types as noted.

38.5.4.14 Device Threshold Control Register (USB0_DTHRCTL)

Offset: 0830h

This register is valid only for device mode in Dedicated FIFO operation (OTG_EN_DED_TX_FIFO = 1). Thresholding is not supported in Slave mode and so this register should not be programmed in Slave mode. For threshold support, the AHB needs to be run at 60MHz or higher.

Figure 38-45. Device Threshold Control Register (USB0_DTHRCTL)

Field	Description	Reset	Access
0:3	Reserved	4'b0	
4	Arbiter Parking Enable (ArbPrkEn) Controls internal DMA arbiter parking for IN endpoints. When thresholding is enabled and this bit is set, the arbiter parks on the IN endpoint for which there is a token received on the USB. This is done to avoid getting into underrun conditions. By default the parking is enabled.	1'b1	R_W
5	Reserved	1'b1	R_W
6:14	Receive Threshold Length (RxThrLen) Specifies Receive thresholding size in DWORDS. Also specifies the amount of data received on the USB before the core can start transmitting on the AHB. The threshold length has to be at least eight DWORDS. The recommended value for ThrLen is to be the same as the programmed AHB Burst Length (GAHBCFG.HBstLen).	9'h8	R_W
15	Receive Threshold Enable (RxThrEn) When set, the function enables thresholding in the receive direction.	1'b0	R_W
16:20	Reserved	5'b0	R_W
21:29	Transmit Threshold Length (TxThrLen) Specifies Transmit thresholding size in DWORDS. Specifies the amount of data in bytes to be in the corresponding endpoint transmit FIFO, before the core can start transmit on the USB. The threshold length has to be at least eight DWORDS. This field controls both isochronous and non-isochronous IN endpoint thresholds. The recommended value for ThrLen is to be the same as the programmed AHB Burst Length (GAHBCFG.HBstLen).	9'h8	R_W
30	ISO IN Endpoints Threshold Enable (ISOThrEn) When set, the function enables thresholding for isochronous IN endpoints.	1'b0	R_W
31	Non-ISO IN Endpoints Threshold Enable (NonISOThrEn) When set, the function enables thresholding for Non Isochronous IN endpoints.	1'b0	R_W

38.5.4.15 Device Control IN Endpoint 0 Control Register (USB0_DIEPCTL0)

This section describes the Control IN Endpoint 0 Control register. Non-zero control endpoints use registers for endpoints 1–15.

Figure 38-46. Device Control IN Endpoint 0 Control Register (USB0_DIEPCTL0)

Field	Description	Reset	Access
0	Endpoint Enable (EPEna) Indicates that data is ready to be transmitted on the Endpoint. The function clears this bit before setting any of the following interrupts on this Endpoint: • Endpoint Disabled • Transfer Completed	1'b0	R_WS_SC

Figure 38-46. Device Control IN Endpoint 0 Control Register (USB0_DIEPCTL0) (Continued)

Field	Description	Reset	Access
1	Endpoint Disable (EPDis) The application sets this bit to stop transmitting data on an Endpoint, even before the transfer for that Endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the Endpoint as disabled. The function clears this bit before setting the Endpoint Disabled Interrupt. The application must set this bit only if Endpoint Enable is already set for this Endpoint.	1'b0	R_WS_SC
2:3	Reserved	2'b0	
4	Set NAK (SNAK) A write to this bit sets the NAK bit for the Endpoint. Using this bit, the application can control the transmission of NAK handshakes on an Endpoint. The function can also set this bit for an Endpoint after a SETUP packet is received on that Endpoint.	1'b0	WO
5	Clear NAK (CNAK) A write to this bit clears the NAK bit for the Endpoint.	1'b0	WO
6:9	TxFIFO Number (TxFNum) • For Shared FIFO operation, this value is always set to 0, indicating that control IN Endpoint 0 data is always written in the Non-Periodic Transmit FIFO. • For Dedicated FIFO operation, this value is set to the FIFO number that is assigned to IN Endpoint 0.	4'h0	R_W
10	STALL Handshake (Stall) The application can only set this bit, and the function clears it, when a SETUP token is received for this Endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority.	1'b0	R_WS_SC
11	Reserved	1'b0	
12:13	Endpoint Type (EPType) Hardcoded to 00 for control.	2'h0	RO
14	NAK Status (NAKSts) Indicates the following: • 1'b0: The function is transmitting non-NAK handshakes based on the FIFO status • 1'b1: The function is transmitting NAK handshakes on this Endpoint. When this bit is set, either by the application or function, the function stops transmitting data, even if there is data available in the TxFIFO. Irrespective of this bit's setting, the function always responds to SETUP data packets with an ACK handshake.	1'b0	RO
15	Reserved	1'b0	
16	USB Active Endpoint (USBActEP) This bit is always set to 1, indicating that control Endpoint 0 is always active in all configurations and interfaces.	1'b1	RO
17:20	Next Endpoint (NextEp) Applies to non-periodic IN endpoints only. Indicates the Endpoint number to be fetched after the data for the current Endpoint is fetched. The function can access this field, even when the Endpoint Enable (EPEna) bit is not set. This field is not valid in Slave mode. **Note:** This field is valid only for Shared FIFO operations.	4'b0	R_W
21:29	Reserved	9'h0	

Figure 38-46. Device Control IN Endpoint 0 Control Register (USB0_DIEPCTL0) (Continued)

Field	Description	Reset	Access
30:31	Maximum Packet Size (MPS) Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical Endpoint. • 2'b00: 64B • 2'b01: 32B • 2'b10: 16B • 2'b11: 8B	2'h0	R_W

38.5.4.16 Device Control OUT Endpoint 0 Control Register (USB0_DOEPCTL0)

This section describes the Control OUT Endpoint 0 Control register. Non-zero control endpoints use registers for endpoints 1–15.

Figure 38-47. Device OUT Endpoint 0 Control Register (USB0_DOEPCTL0)

Field	Description	Reset	Access
0	Endpoint Enable (EPEna) Indicates that the application has allocated the memory to start receiving data from the USB. The function clears this bit before setting any of the following interrupts on this Endpoint: • SETUP Phase Done • Endpoint Disabled • Transfer Completed **Note:** In DMA mode, this bit must be set for the function to transfer SETUP data packets into memory.	1'b0	R_WS_SC
1	Endpoint Disable (EPDis) The application cannot disable control OUT Endpoint 0.	1'b0	RO
2:3	Reserved	2'b0	
4	Set NAK (SNAK) A write to this bit sets the NAK bit for the Endpoint. Using this bit, the application can control the transmission of NAK handshakes on an Endpoint. The function can also set bit on a Transfer Completed interrupt, or after a SETUP is received on the Endpoint.	1'b0	WO
5	Clear NAK (CNAK) A write to this bit clears the NAK bit for the Endpoint.	1'b0	WO
6:9	Reserved	4'h0	
10	STALL Handshake (Stall) The application can only set this bit, and the function clears it, when a SETUP token is received for this Endpoint. If a NAK bit or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the function always responds to SETUP data packets with an ACK handshake.	1'b0	R_WS_SC
11	Snoop Mode (Snp) This bit configures the Endpoint to Snoop mode. In Snoop mode, the function does not check the correctness of OUT packets before transferring them to application memory.	1'b0	R_W
12:13	Endpoint Type (EPType) Hardcoded to 2'b00 for control.	2'h0	RO

Figure 38-47. Device OUT Endpoint 0 Control Register (USB0_DOEPCTL0) (Continued)

Field	Description	Reset	Access
14	NAK Status (NAKSts) Indicates the following: 0 The function is transmitting non-NAK handshakes based on the FIFO status. 1 The function is transmitting NAK handshakes on this Endpoint. When either the application or the function sets this bit, the function stops receiving data, even if there is space in the RxFIFO to accommodate the incoming packet. Irrespective of this bit's setting, the function always responds to SETUP data packets with an ACK handshake.	1'b0	RO
15	Reserved	1'b0	
16	USB Active Endpoint (USBActEP) This bit is always set to 1, indicating that a control Endpoint 0 is always active in all configurations and interfaces.	1'b1	RO
17:29	Reserved	13'h0	
30:31	Maximum Packet Size (MPS) The maximum packet size for control OUT Endpoint 0 is the same as what is programmed in control IN Endpoint 0. 00 64B 01 32B 10 16B 11 8B	2'h0	RO

38.5.4.17 Device Endpoint n Control Register (USB0_DIEPCTLn/DOEPCTLn)

Endpoint_number: $1 \leq n \leq 2$

Offset for IN endpoints: 900h + (Endpoint_number * 20h)

Offset for OUT endpoints: B00h + (Endpoint_number * 20h)

The application uses this register to control the behavior of each logical Endpoint other than Endpoint 0.

Figure 38-48. Device Endpoint n Control Register (USB0_DIEPCTLn/DOEPCTLn)

Field	Description	Reset	Access
0	Endpoint Enable (EPEna) Applies to IN and OUT endpoints. For IN endpoints, this bit indicates that data is ready to be transmitted on the Endpoint. For OUT endpoints, this bit indicates that the application has allocated the memory to start receiving data from the USB. The function clears this bit before setting any of the following interrupts on this Endpoint: • SETUP Phase Done (OUT only) • Endpoint Disabled • Transfer Completed **Note:** For control OUT endpoints in DMA mode, this bit must be set to be able to transfer SETUP data packets in memory.	1'b0	R_WS_SC
1	Endpoint Disable (EPDis) Applies to IN and OUT endpoints. The application sets this bit to stop transmitting/receiving data on an Endpoint, even before the transfer for that Endpoint is complete. The application must wait for the Endpoint Disabled interrupt before treating the Endpoint as disabled. The function clears this bit before setting the Endpoint Disabled interrupt. The application must set this bit only if Endpoint Enable is already set for this Endpoint.	1'b0	R_WS_SC

Figure 38-48. Device Endpoint n Control Register (USB0_DIEPCTLn/DOEPCTLn) (Continued)

Field	Description	Reset	Access
2	Set DATA1 PID (SetD1PID) Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the Endpoint Data PID (DPID) field in this register to DATA1. Set Odd (micro)frame (SetOddFr) Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd (micro)frame (EO_FrNum) field to odd (micro)frame.	1'b0	WO
3	Set DATA0 PID (SetD0PID) Applies to interrupt/bulk IN and OUT endpoints only. Writing to this field sets the Endpoint Data PID (DPID) field in this register to DATA0. Set Even (micro)frame (SetEvenFr) Applies to isochronous IN and OUT endpoints only. Writing to this field sets the Even/Odd (micro)frame (EO_FrNum) field to even (micro) frame.	1'b0	WO
4	Set NAK (SNAK) Applies to IN and OUT endpoints. A write to this bit sets the NAK bit for the Endpoint. Using this bit, the application can control the transmission of NAK handshakes on an Endpoint. The function can also set this bit for OUT endpoints on a Transfer Completed interrupt, or after a SETUP is received on the Endpoint.	1'b0	WO
5	Clear NAK (CNAK) Applies to IN and OUT endpoints. A write to this bit clears the NAK bit for the Endpoint.	1'b0	WO
6:9	TxFIFO Number (TxFNum) *Shared FIFO Operation* —non-periodic endpoints must set this bit to zero. Periodic endpoints must map this to the corresponding Periodic TxFIFO number. • 4h0:Non-Periodic TxFIFO • Others: Specified Periodic TxFIFO.number **Note:** An interrupt IN Endpoint can be configured as a nonperiodic Endpoint for applications like mass storage. The function treats an IN Endpoint as a non-periodic Endpoint if the TxFNum field is set to 0.	4'h0	R_W
10	STALL Handshake (Stall) Applies to non-control, non-isochronous IN and OUT endpoints only. The application sets this bit to stall all tokens from the USB Host to this Endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Only the application can clear this bit, never the function. Applies to control endpoints only. The application can only set this bit, and the function clears it, when a SETUP token is received for this Endpoint. If a NAK bit, Global Non-periodic IN NAK, or Global OUT NAK is set along with this bit, the STALL bit takes priority. Irrespective of this bit's setting, the function always responds to SETUP data packets with an ACK handshake.	1'b0	R_W R_WS_SC
11	Snoop Mode (Snp) Applies to OUT endpoints only. This bit configures the Endpoint to Snoop mode. In Snoop mode, the function does not check the correctness of OUT packets before transferring them to application memory.	1'b0	R_W
12:13	Endpoint Type (EPType) Applies to IN and OUT endpoints. This is the transfer type supported by this logical Endpoint. 00 Control 01 Isochronous 10 Bulk 11 Interrupt	2'h0	R_W

Figure 38-48. Device Endpoint n Control Register (USB0_DIEPCTLn/DOEPCTLn) (Continued)

Field	Description	Reset	Access
14	NAK Status (NAKSts) Applies to IN and OUT endpoints. Indicates the following: 0 The function is transmitting non-NAK handshakes based on the FIFO status. 1 The function is transmitting NAK handshakes on this Endpoint. When either the application or the function sets this bit: • The function stops receiving any data on an OUT Endpoint, even if there is space in the RxFIFO to accommodate the incoming packet. • For non-isochronous IN endpoints: The function stops transmitting any data on an IN Endpoint, even if there data is available in the TxFIFO. • For isochronous IN endpoints: The function sends out a zero-length data packet, even if there data is available in the TxFIFO. Irrespective of this bit's setting, the function always responds to SETUP data packets with an ACK handshake.	1'b0	RO
15	Endpoint Data PID (DPID) Applies to interrupt/bulk IN and OUT endpoints only. Contains the PID of the packet to be received or transmitted on this Endpoint. The application must program the PID of the first packet to be received or transmitted on this Endpoint, after the Endpoint is activated. Applications use the SetD1PID and SetD0PID fields of this register to program either DATA0 or DATA1 PID. 0 DATA0 1 DATA1 Even/Odd (Micro)Frame (EO_FrNum) Applies to isochronous IN and OUT endpoints only. Indicates the (micro)frame number in which the function transmits/receives isochronous data for this Endpoint. The application must program the even/odd (micro) frame number in which it intends to transmit/receive isochronous data for this Endpoint using the SetEvnFr and SetOddFr fields in this register. 0 Even (micro)frame 1 Odd (micro)frame	1'b0	RO
16	USB Active Endpoint (USBActEP) Applies to IN and OUT endpoints. Indicates whether this Endpoint is active in the current configuration and interface. The function clears this bit for all endpoints (other than EP 0) after detecting a USB reset. After receiving the SetConfiguration and SetInterface commands, the application must program Endpoint registers accordingly and set this bit.	1'b0	R_W_SC
17:20	Next Endpoint (NextEp) Applies to non-periodic IN endpoints only. Indicates the Endpoint number to be fetched after the data for the current Endpoint is fetched. The function can access this field, even when the Endpoint Enable (EPEna) bit is low. This field is not valid in Slave mode operation. **Note:** This field is valid only for Shared FIFO operations.	4'b0	R_W
21:31	Maximum Packet Size (MPS) Applies to IN and OUT endpoints. The application must program this field with the maximum packet size for the current logical Endpoint. This value is in bytes.	11'h0	R_W

38.5.4.18 Device Endpoint n Interrupt Register (USB0_DIEPINTn/DOEPINTn)

Endpoint_number: $0 \leq n \leq 2$

Offset for IN endpoints: 908h + (Endpoint_number * 20h)

Offset for OUT endpoints: B08h + (Endpoint_number * 20h)

This register indicates the status of an Endpoint with respect to USB- and AHB-related events. The application must read this register when the OUT Endpoints Interrupt bit or IN Endpoints Interrupt bit of the Interrupt register (GINTSTS.OEPInt or GINTSTS.IEPInt, respectively) is set. Before the application can read this register, it must first read the Device All Endpoints Interrupt (DAINT) register to get the exact Endpoint number for the Device Endpoint-n Interrupt register. The application must clear the appropriate bit in this register to clear the corresponding bits in the DAINT and GINTSTS registers.

Figure 38-49. Device Endpoint n Interrupt Register (USB0_DIEPINTn/DOEPINTn)

Field	Description	Reset	Access
0:22	Reserved	23h0'	
23	Fifo Under run (TxfifoUndrn) Applies to IN endpoints only. This bit is valid only if OTG_EN_DED_TX_FIFO = 1 and thresholding is enabled. The function generates this interrupt when it see a transmit FIFO underrun condition for this Endpoint. OUT Packet Error (OutPktErr) Applies to OUT endpoints only. This interrupt is valid only when OTG_EN_DED_TX_FIFO = 1 and thresholding is enabled. This interrupt is asserted when the function sees an overflow or a CRC error for non-ISOC OUT packet.	1'b0	R_W
24	Transmit FIFO Empty (TxFEmp) This bit is valid only for IN Endpoints This interrupt is asserted when the TxFIFO for this Endpoint is either half or completely empty. The half or completely empty status is determined by the TxFIFO Empty Level bit in the AHB Configuration register (GAHBCFG.NPTxFEmpLvl).	1'b0	R_W
25	IN Endpoint NAK Effective (INEPNakEff) Applies to periodic IN endpoints only. Indicates that the IN Endpoint NAK bit set by the application has taken effect in the function. This bit can be cleared when the application clears the IN Endpoint NAK by writing to DIEPCTLn.CNAK. This interrupt indicates that the function has sampled the NAK bit set (either by the application or by the function). This interrupt does not necessarily mean that a NAK handshake is sent on the USB. A STALL bit takes priority over a NAK bit. Back-to-Back SETUP Packets Received (Back2BackSETup) Applies to Control OUT endpoints only. This bit indicates that the function has received more than three back-to-back SETUP packets for this particular Endpoint.	1'b0 1'b0	RO R_W
26	IN Token Received with EP Mismatch (INTknEPMis) Applies to non-periodic IN endpoints only. Indicates that the data in the top of the non-periodic TxFIFO belongs to an Endpoint other than the one for which the IN token was received. This interrupt is asserted on the Endpoint for which the IN token was received. For OUT endpoints, this bit is Reserved	1'b0	R_SS_WC

Figure 38-49. Device Endpoint n Interrupt Register (USB0_DIEPINTn/DOEPINTn) (Continued)

Field	Description	Reset	Access
27	IN Token Received When TxFIFO is Empty (INTknTXFEmp) Applies to non-periodic IN endpoints only. Indicates that an IN token was received when the associated TxFIFO (periodic/non-periodic) was empty. This interrupt is asserted on the Endpoint for which the IN token was received. OUT Token Received When Endpoint Disabled (OUTTknEPdis) Applies only to control OUT endpoints. Indicates that an OUT token was received when the Endpoint was not yet enabled. This interrupt is asserted on the Endpoint for which the OUT token was received.	1'b0	R_SS_WC
28	Timeout Condition (TimeOUT) Applies to non-isochronous IN endpoints in shared FIFO operation only. Indicates that the function has detected a timeout condition on the USB for the last IN token on this Endpoint. SETUP Phase Done (SetUp) Applies to control OUT endpoints only. Indicates that the SETUP phase for the control Endpoint is complete and no more back-to-back SETUP packets were received for the current control transfer. On this interrupt, the application can decode the received SETUP data packet.	1'b0	R_SS_WC
29	AHB Error (AHBErr) Applies to IN and OUT endpoints. This is generated only in Internal DMA mode when there is an AHB error during an AHB read/write. The application can read the corresponding Endpoint DMA address register to get the error address.	1'b0	R_SS_WC
30	Endpoint Disabled Interrupt (EPDisbld) Applies to IN and OUT endpoints. This bit indicates that the Endpoint is disabled per the application's request.	1'b0	R_SS_WC
31	Transfer Completed Interrupt (XferCompl) Applies to IN and OUT endpoints. Indicates that the programmed transfer is complete on the AHB as well as on the USB, for this Endpoint.	1'b0	R_SS_WC

38.5.4.19 Device Endpoint 0 Transfer Size Register (USB0_DIEPTSIZ0/DOEPTSIZ0))

Offset for IN endpoints: 910h

Offset for OUT endpoints: B10h

The application must modify this register before enabling Endpoint 0. Once Endpoint 0 is enabled using Endpoint Enable bit of the Device Control Endpoint 0 Control registers (DIEPCTL0.EPEna/DOEPCTL0.EPEna), the function modifies this register. The application can only read this register once the function has cleared the Endpoint Enable bit. Non-zero endpoints use the registers for endpoints 1–15.

Figure 38-50. Device IN Endpoint 0 Transfer Size Register (USB0_DIEPTSIZ0)

Field	Description	Reset	Access
0:11	Reserved	12'h0	
12	Packet Count (PktCnt) Indicates the total number of USB packets that constitute the Transfer Size amount of data for Endpoint 0. This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO.	1'b0	R_W

Figure 38-50. Device IN Endpoint 0 Transfer Size Register (USB0_DIEPTSIZ0) (Continued)

Field	Description	Reset	Access
13:24	Reserved	12'h0	
25:31	Transfer Size (XferSize) Indicates the transfer size in bytes for Endpoint 0. The function interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the Endpoint, to be interrupted at the end of each packet. The function decrements this field every time a packet from the external memory is written to the TxFIFO.	7'h0	R_W

Figure 38-51. Device OUT Endpoint 0 Transfer Size Register (USB0_DOEPTSIZ0)

Field	Description	Reset	Access
0	Reserved	1'b0	
1:2	SETUP Packet Count (SUPCnt) This field specifies the number of back-to-back SETUP data packets the Endpoint can receive. • 2'b01: 1 packet • 2'b10: 2 packets • 2'b11: 3 packets	2'h0	R_W
3:11	Reserved	9'h0	
12	Packet Count (PktCnt) This field is decremented to zero after a packet is written into the RxFIFO.	1'b0	R_W
13:24	Reserved	12'h0	
25:31	Transfer Size (XferSize) Indicates the transfer size in bytes for Endpoint 0. The function interrupts the application only after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the Endpoint, to be interrupted at the end of each packet. The function decrements this field every time a packet is read from the RxFIFO and written to the external memory.	7'h0	R_W

38.5.4.20 Device Endpoint n Transfer Size Register (USB0_DIEPTSIZn/DOEPTSIZn)

Endpoint_number: $1 \leq n \leq 15$

Offset for IN endpoints: 910h + (Endpoint_number * 20h)

Offset for OUT endpoints: B10h + (Endpoint_number * 20h)

The application must modify this register before enabling the Endpoint. Once the Endpoint is enabled using Endpoint Enable bit of the Device Endpoint-n Control registers (DIEPCTLn.EPEna/DOEPCTLn.EPEna), the function modifies this register. The application can only read this register once the function has cleared the Endpoint Enable bit.
This register is used only for endpoints other than Endpoint 0.

Figure 38-52. Device Endpoint n Transfer Size Register (USB0_DIEPTSIZn/DOEPTSIZn)

Field	Description	Reset	Access
0	Reserved	1'b0	
1:2	Multi Count (MC) Applies to IN endpoints only. For periodic IN endpoints, this field indicates the number of packets that must be transmitted per microframe on the USB. The function uses this field to calculate the data PID for isochronous IN endpoints. • 2'b01: 1 packet • 2'b10: 2 packets • 2'b11: 3 packets For non-periodic IN endpoints, this field is valid only in Internal DMA mode. It specifies the number of packets the function should fetch for an IN Endpoint before it switches to the Endpoint pointed to by the Next Endpoint field of the Device Endpoint-n Control register (DIEPCTLn.NextEp). Received Data PID (RxDPID) Applies to isochronous OUT endpoints only. This is the data PID received in the last packet for this Endpoint. • 2'b00: DATA0 • 2'b01: DATA1 • 2'b10: DATA2 • 2'b11: MDATASETUP Packet Count (SUPCnt) Applies to control OUT Endpoints only. This field specifies the number of back-to-back SETUP data packets the Endpoint can receive. • 2'b01: 1 packet • 2'b10: 2 packets • 2'b11: 3 packets	2'b0	R_W RO R_W R_W
3:12	Packet Count (PktCnt) Indicates the total number of USB packets that constitute the Transfer Size amount of data for this Endpoint. The power-on value is specified for Width of Packet Counters during functionConsultant configuration (parameter OTG_PACKET_COUNT_WIDTH). • IN Endpoints: This field is decremented every time a packet (maximum size or short packet) is read from the TxFIFO. • OUT Endpoints: This field is decremented every time a packet (maximum size or short packet) is written to the RxFIFO.	10'h0	R_W
13:31	Transfer Size (XferSize) This field contains the transfer size in bytes for the current Endpoint. The power-on value is specified for Width of Transfer Size Counters during functionConsultant configuration (parameter OTG_TRANS_COUNT_WIDTH). The function only interrupts the application after it has exhausted the transfer size amount of data. The transfer size can be set to the maximum packet size of the Endpoint, to be interrupted at the end of each packet. • IN Endpoints: The function decrements this field every time a packet from the external memory is written to the TxFIFO. • OUT Endpoints: The function decrements this field every time a packet is read from the RxFIFO and written to the external memory.	19'h0	R_W

38.5.4.21 Device Endpoint n DMA Address Register (USB0_DIEPDMAn/DOEPDMAn)

Endpoint_number: $0 \leq n \leq 2$

Offset for IN endpoints: 914h + (Endpoint_number * 20h)

Offset for OUT endpoints: B14h + (Endpoint_number * 20h)

Figure 38-53. Device Endpoint n DMA Address Register (USB0_DIEPDMAn/DOEPDMAn)

Field	Description	Reset	Access
0:31	DMA Address (DMAAddr) Holds the start address of the external memory for storing or fetching Endpoint data. This register is incremented on every AHB transaction. **Note:** For control endpoints, this address stores control OUT data packets as well as SETUP transaction data packets. If more than three SETUP packets are received back-to-back, the, the SETUP data packet in the memory is overwritten.	32'h0	R_W

38.5.4.22 Device IN Endpoint Transmit FIFO Status Register (USB0_DTXFSTSn)

Endpoint_number: $0 \leq n \leq 15$

Offset for IN endpoints: 918h + (Endpoint_number * 20h)

This read-only register contains the free space information for the Device IN Endpoint TxFIFO.

Figure 38-54. Device IN Endpoint Transmit FIFO Status Register (USB0_DTXFSTSn

Field	Description	Reset	Access
0:15	Reserved	1'b0	
16:31	IN Endpoint TxFIFO Space Avail (INEPTxFSpcAvail) Indicates the amount of free space available in the Endpoint TxFIFO. Values are in terms of 32-bit words. • 16'h0: Endpoint TxFIFO is full • 16'h1: 1 word available • 16'h2: 2 words available • 16'h n: n words available (where $0 \leq n \leq 32,768$) • 16'h8000: 32,768 words available • Others: Reserved	User selected	RO

38.5.5 Clock Gating Registers

There is only one register in this group as follows.

38.5.5.1 Clock Gating Control Register (USB0_PCGCTL)

Offset: E00h

This register is available in Host and Device modes. The PwrClmp bit is available only if the OTG_EN_PWROPT parameter is set to 1 during function configuration. The application can use this register to control clock gating features.

Because the CSR module is turned off during power-down, this registers is implemented in the AHB Slave BIU module.

Figure 38-55. Clock Gating Control Register (USB0_PCGCTL)

Field	Description	Reset	Access
0:29	Reserved	27'h0	
30	Gate Hclk (GateHclk) The application sets this bit to gate hclk to modules other than the AHB Slave and Master and wakeup logic when the USB is suspended or the session is not valid. The application clears this bit when the USB is resumed or a new session starts.	1'b0	R_W
31	Stop Pclk (StopPclk) The application sets this bit to stop the PHY clock (phy_clk) when the USB is suspended, the session is not valid, or the Device is disconnected. The application clears this bit when the USB is resumed or a new session starts.	1'b0	R_W

38.6 USB Programming Models Overview

The USB programming models describes the programming requirements for the USB function in Host and Device modes. Each significant programming feature of the function is discussed in a separate section.

The description uses the format *register_name* (less USB0_) to reference particular registers, and the format *register_name*(less USB0_).*field* to reference particular register fields. Field names are given in each register description. For detailed information on registers, see *Register Descriptions* on page 1340.

38.7 Initialization

The application must perform the initialization sequence. If a cable is connected during power-up, the Current Mode of Operation bit in the Interrupt register (GINTSTS.CurMod) reflects the mode. The USB function enters Host mode when an "A" plug is connected, or Device mode when a "B" plug is connected.

This section explains the initialization of the USB function after power-on. The application must follow the initialization sequence irrespective of Host or Device mode operation.

1. Enable the USB interface.
   - Set SDR0_PFC1[17]
   - Set GPIO signals
   - Reset SDR0_SRST[30], SDR0_SRST[10], and SDR0_SRST[17]
2. Clear all USB interrupts by writing GINSTST.
3. Program the following fields in the Global AHB Configuration (GAHBCFG) register.
   - DMA Mode bit
   - AHB Burst Length field
   - Global Interrupt Mask bit = 1
   - RxFIFO Non-Empty (GINTSTS.RxFLvl) (applicable only when the function is operating in Slave mode)
   - non-periodic TxFIFO Empty Level (can be enabled only when the function is operating in Slave mode as a Host or as a Device in Shared FIFO operation.)
   - Periodic TxFIFO Empty Level (can be enabled only when the function is operating in Slave mode)
4. Program the following fields in GUSBCFG register.

- HNP Capable bit
- SRP Capable bit
- ULPI Selection bit
- HS/FS TimeOUT Time-Out Calibration field
- USB Turnaround Time field

5. The software must unmask the following bits in the GINTMSK register.
   - OTG Interrupt Mask
   - Mode Mismatch Interrupt Mask
6. The software can read the GINTSTS.CurMod bit to determine whether the USB function is operating in Host or Device mode. The software then does either Host initialization or Device initialization.

38.7.1 Host Initialization

To initialize the function as a Host, the application must perform the following steps:

1. Program GINTMSK.PrtInt to unmask.
2. Program the HCFG register to select full-speed Host or high-speed Host.
3. Program the HPRT.PrtPwr bit to 1'b1. This drives V_{BUS} on the USB.
4. Wait for the HPRT0.PrtConnDet interrupt. This indicates that a Device is connect to the port.
5. Program the HPRT.PrtRst bit to 1'b1. This starts the reset process.
6. Wait at least 10 ms for the reset process to complete.
7. Program the HPRT.PrtRst bit to 1'b0.
8. Wait for the HPRT.PrtEnChng interrupt.
9. Read the HPRT.PrtSpd field to get the enumerated speed.
10. Program the HFIR register with a value corresponding to the selected PHY clock.
    (At this point, the Host is up and running and the port register begins to report Device disconnects, etc. The port is active with SOFs occurring down the enabled port.)
11. Program the RXFSIZE register to select the size of the receive FIFO.
12. Program the NPTXFSIZE register to select the size and the start address of the Non-periodic Transmit FIFO for non-periodic transactions.
13. Program the HPTXFSIZ register to select the size and start address of the Periodic Transmit FIFO for periodic transactions.

To communicate with devices, the system software must initialize and enable at least one channel.

38.7.2 Device Initialization

The application must perform the following steps to initialize the USB function as a Device on power-up or after a mode change from Host to Device.

1. Program the following fields in the DCFG register.

- Device Speed
- Non-Zero Length Status OUT Handshake
- Periodic Frame Interval

2. Program the Device threshold control register. This is required only if you are using DMA mode and you are planning to enable thresholding.
3. Program the GINTMSK register to unmask the following interrupts.
   - USB Reset
   - Enumeration Done
   - Early Suspend
   - USB Suspend
   - SOF
4. Wait for the GINTSTS.USBReset interrupt, which indicates a reset has been detected on the USB and lasts for about 10ms. On receiving this interrupt, the application must perform the steps listed in *Initialization on USB Reset* on page 1444.
5. Wait for the GINTSTS.EnumerationDone interrupt. This interrupt indicates the end of reset on the USB. On receiving this interrupt, the application must read the DSTS register to determine the enumeration speed and perform the steps listed in *Initialization on Enumeration Completion* on page 1445.

At this point, the Device is ready to accept SOF packets and perform control transfers on control Endpoint 0.

38.8 Modes of Operation

The application can operate the function either in DMA mode, where the function fetches the data to be transmitted or updates the received data on the AHB, or in Slave mode, where the application initiates transfers for data fetch and store. The application cannot operate the function using DMA and Slave modes simultaneously.

38.8.1 DMA Mode

The application is using the internal DMA in this mode. With the internal DMA, the USB Host uses the AHB Master interface for transmit packet data fetch (AHB to USB) and receive data update (USB to AHB). The AHB Master uses the programmed DMA address (HCDMAn register in Host mode and DIEPDMAn/DOEPDMAn register in Device mode) to access the data buffers.

38.8.1.1 Transfer-Level Operation

In DMA mode, the application is interrupted only after the programmed transfer size is transmitted or received (provided the USB function detects no NAK/NYET/Timeout/Error response in Host mode, or Timeout/CRC Error in Device mode). The application must handle all transaction errors. In Device mode, when dedicated FIFOs are chosen (OTG_EN_DED_TX_FIFO = 1), then all the USB errors are handled by the function itself.

38.8.1.2 Transaction-Level Operation

This mode is similar to transfer-level operation, with the programmed transfer size equal to one packet size (maximum size or short packet).

38.8.2 Slave Mode

In Slave mode, the application can operate the USB function either in transaction-level (packet-level) operation or in pipelined transaction-level operation.

38.8.2.1 Transaction-Level Operation

The application handles one data packet at a time per channel/Endpoint in transaction-level operations. Based on the handshake response received on the USB, the application determines whether to retry the transaction or proceed with the next, until the end of the transfer. The application is interrupted on completion of every packet. The application performs transaction-level operations for a channel/Endpoint for a transmission (Host: OUT/Device: IN) or reception (Host: IN/Device: OUT).

Host Mode

For an OUT transaction, the application enables the channel and writes the data packet into the corresponding (Periodic or Non-periodic) transmit FIFO. The USB function automatically writes the channel number into the corresponding (Periodic or Non-periodic) Request Queue, along with the last DWORD write of the packet. For an IN transaction, the application enables the channel and the USB function automatically writes the channel number into the corresponding Request queue. The application must wait for the packet received interrupt, then empty the packet from the receive FIFO.

Device Mode

For an IN transaction, the application enables the Endpoint, writes the data packet into the corresponding transmit FIFO, and waits for the packet completion interrupt from the function. For an OUT transaction, the application enables the Endpoint, waits for the packet received interrupt from the function, then empties the packet from the receive FIFO.

The application has to finish writing one complete packet before switching to a different channel/Endpoint FIFO. Violating this rule results in an error.

Note: If the DMA to PLB4 (DMA2P40) is used to move the packet data in and out of the FIFOs, then the application must wait until the DMA2P40 has completed the transfer before reprogramming any registers in the USB 2.0 OTG.

Figure 38-56. Transmit Transaction-Level Operation in Slave Mode

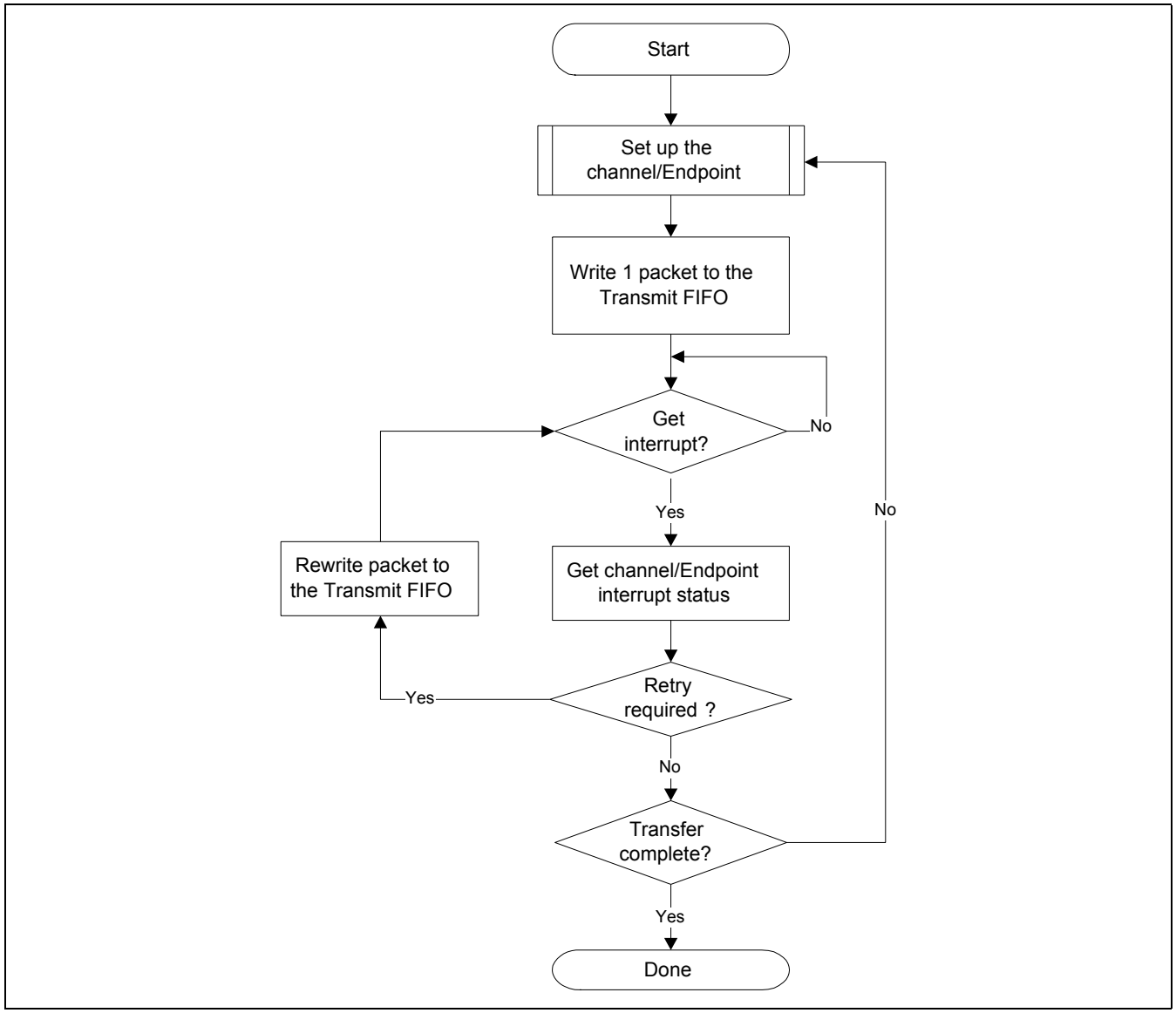
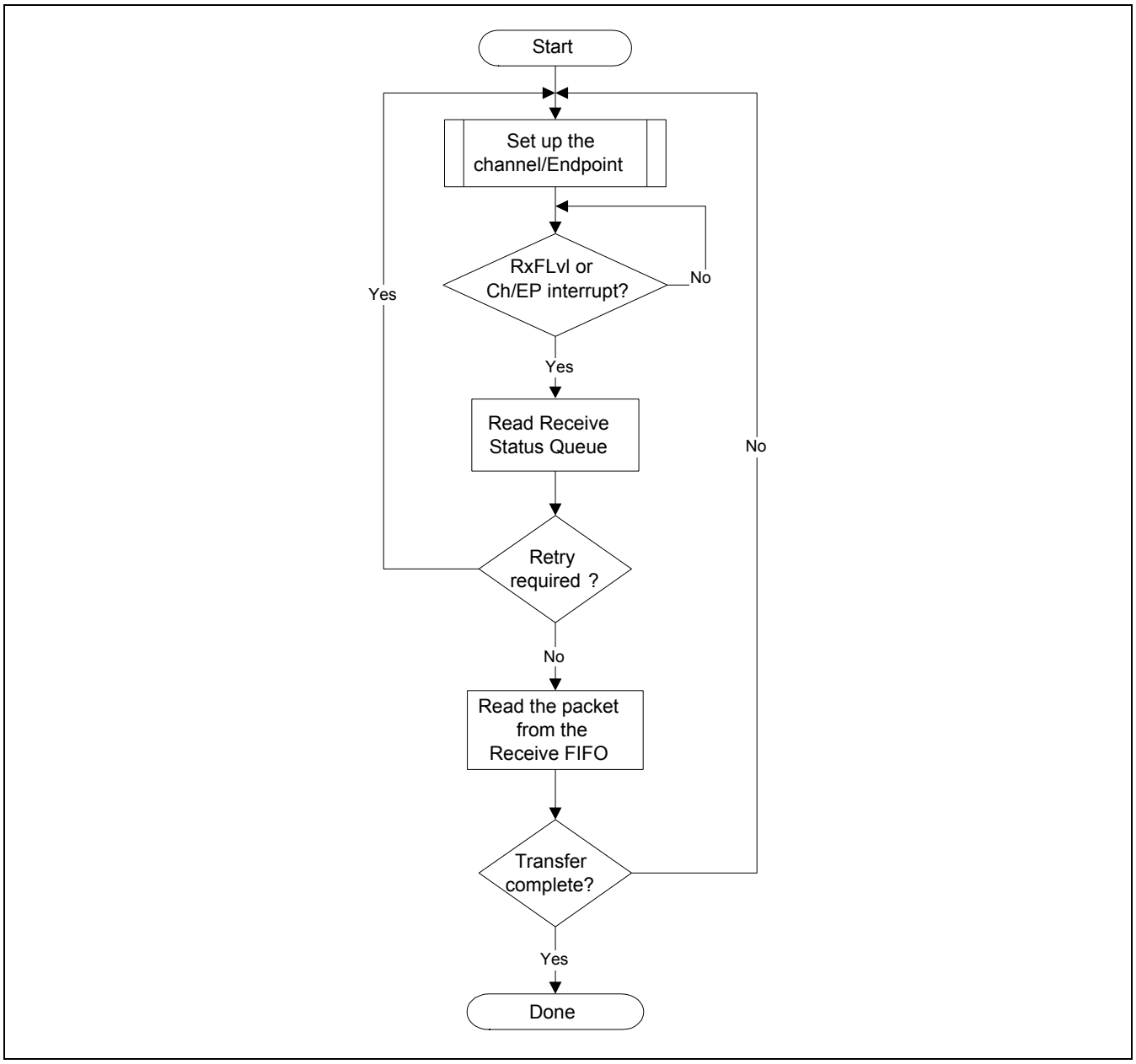


Figure 38-57. Receive Transaction-Level Operation in Slave Mode



38.8.2.2 Pipelined Transaction-Level Operation

The application can pipeline more than one transaction (IN or OUT) with pipelined transaction-level operation, which is analogous to Transfer mode in DMA mode. In pipelined transaction-level operation, the application can program the function to perform multiple transactions. The advantage of this mode of operation compared to transaction-level operation is that the application is not interrupted on packet basis.

Host Mode

For an OUT transaction, the application sets up a transfer and enables the channel. The application can write multiple packets back-to-back for the same channel into the transmit FIFO, based on the space availability. It can also pipeline OUT transactions for multiple channels by writing into the HCHARn register, followed by a packet write to that channel. The function writes the channel number, along with the last DWORD write for the packet, into the Request queue and schedules transactions on the USB in the same order.

For an IN transaction, the application sets up a transfer and enables the channel, and the USB function writes the channel number into the Request queue. The application can schedule IN transactions on multiple channels, provided space is available in the Request queue. The function initiates an IN token on the USB only when there is enough space to receive at least of one maximum-packet-size packet of the channel in the top of the Request queue.

Device Mode

For an IN transaction, the application sets up a transfer and enables the Endpoint. The application can write multiple packets back-to-back for the same Endpoint into the transmit FIFO, based on available space. It can also pipeline IN transactions for multiple channels by writing into the DIEPCTLn register followed by a packet write to that Endpoint. The function writes the Endpoint number, along with the last DWORD write for the packet into the Request queue. The function transmits the data in the transmit FIFO when an IN token is received on the USB.

For an OUT transaction, the application sets up a transfer and enables the Endpoint. The function receives the OUT data into the receive FIFO, when it has available space. As the packets are received into the FIFO, the application must empty data from it.

Note: If the DMA to PLB4 (DMA2P40) is used to move the packet data in and out of the FIFOs, then the application must wait until the DMA2P40 has completed the transfer before reprogramming any registers in the USB 2.0 OTG.

From this point on in this chapter, the terms “Pipelined Transaction mode” and “Transfer mode” are used interchangeably.

38.9 Host Programming Model

The following sections describe the Host programming model.

38.9.1 Channel Initialization

The application must initialize one or more channels before it can communicate with connected devices. To initialize and enable a channel, the application must perform the following steps.

1. Program the GINTMSK register to unmask the following:
   - Channel Interrupt
   - Non-periodic Transmit FIFO Empty for OUT transactions (applicable for Slave mode that operates in pipelined transaction-level with the Packet Count field programmed with more than one).
   - Non-periodic Transmit FIFO Half-Empty for OUT transactions (applicable for Slave mode that operates in pipelined transaction-level with the Packet Count field programmed with more than one).
2. Program the HAINTMSK register to unmask the selected channels' interrupts.
3. Program the HCINTMSK register to unmask the transaction-related interrupts of interest given in the Host Channel Interrupt register.

4. Program the selected channel's HCTSIZn register with the total transfer size, in bytes, and the expected number of packets, including short packets. The application must program the PID field with the initial data PID (to be used on the first OUT transaction or to be expected from the first IN transaction).
5. Program the selected channels' HCSPLTn register(s) with the hub and port addresses (split transactions only).
6. Program the selected channels' HCDMAn register(s) with the buffer start address (DMA mode only).
7. Program the HCCHARn register of the selected channel with the Device's Endpoint characteristics, such as type, speed, direction, and so forth. (The channel can be enabled by setting the Channel Enable bit to 1'b1 only when the application is ready to transmit or receive any packet).

Repeat steps 1–7 for other channels.

38.9.2 Halting a Channel

The application can disable any channel by programming the HCCHARn register with the HCCHARn.ChDis and HCCHARn.ChEna bits set to 1'b1. This enables the USB Host to flush the posted requests (if any) and generates a Channel Halted interrupt. The application must wait for the HCINTn.ChHltd interrupt before reallocating the channel for other transactions. The USB Host does not interrupt the transaction that has been already started on USB.

In Slave mode operation, before disabling a channel, the application must ensure that there is at least one free space available in the Non-periodic Request Queue (when disabling a non-periodic channel) or the Periodic Request Queue (when disabling a periodic channel). The application can simply flush the posted requests when the Request queue is full (before disabling the channel), by programming the HCCHARn register with the HCCHARn.ChDis bit set to 1'b1, and the HCCHARn.ChEna bit reset to 1'b0.

To disable a channel in DMA mode operation, the application need not check for space in the Request queue. The USB Host checks for space in which to write the Disable request on the disabled channel's turn during arbitration. Meanwhile, all posted requests are dropped from the Request queue when the HCCHARn.ChDis bit is set to 1'b1.

The application is expected to disable a channel on any of the following conditions:

1. When a HCINTn.XferCompl interrupt is received during a non-periodic IN transfer or high-bandwidth interrupt IN transfer (Slave mode only)
2. When a HCINTn.STALL, HCINTn.XactErr, HCINTn.BblErr, or HCINTn.DataTglErr interrupt is received for an IN or OUT channel (Slave mode only). For high-bandwidth interrupt INs in Slave mode, once the application has received a DataTglErr interrupt it must disable the channel and wait for a Channel Halted interrupt. The application must be able to receive other interrupts (DataTglErr, Nak, Data, XactErr, BabbleErr) for the same channel before receiving the halt.
3. When a GINTSTS.DisconnInt (Disconnect Device) interrupt is received. (The application is expected to disable all enabled channels in Slave and DMA modes.)
4. When the application aborts a transfer before normal completion (Slave and DMA modes).

38.9.3 Ping Protocol

When the USB Host operates in high speed, the application must initiate the ping protocol when communicating with high-speed bulk or control (Data and Status stage) OUT endpoints. The application must initiate the ping protocol when it receives a NAK/NYET/XactErr interrupt. When the USB Host receives one of the above responses, it does not continue any transaction for a specific Endpoint, drops all posted or fetched OUT requests (from the Request queue), and flushes the corresponding data (from the transmit FIFO).

In Slave mode, the application can send a ping token either by setting the HCTSIZn.DoPng bit before enabling the channel or by just writing the HCTSIZn register with DoPng bit set when the channel is already enabled. This enables the USB Host to write a ping request entry to the Request queue. The application must wait for the response to the ping token (a NAK, ACK, or XactErr interrupt) before continuing the transaction or sending another ping token. The application can continue the data transaction only after receiving an ACK from the OUT Endpoint for the requested ping. The channel-specific interrupt service routine for the ping protocol in Slave mode is shown in *Table 38-58* on page 1399.

n DMA mode operation, the application can start a ping protocol transfer by setting the HCTSIZn.DoPng bit before enabling the channel. The USB Host continues sending ping tokens until receiving an ACK, then switches automatically to the data transaction.

Figure 38-58. ISR for Ping Protocol in Slave Mode

Ping Protocol for High Speed Bulk/Control OUT Endpoints

```
Unmask (ACK/NAK/XactErr/ChHltd/STALL)
if (ACK)
{
Reset Error Count
Re-initialize Channel (Do data transactions)
{
else if (NAK)
{
Reset Error Count Send Ping
}
else if (STALL)
{
Disable Channel
}
else if (XactErr)
{
Increment Error Count
    if (Error_count < 3)
    {
    Send Ping
    }
    else
    {
    Disable Channel
    }
}
else if (ChHltd)
{
De-allocate Channel
}
```

38.9.4 Sending a Zero-Length Packet

To send a zero-length data packet, the application must initialize an OUT channel as follows.

1. Program the HCTSIZn register of the selected channel with a correct PID, XferSize = 0, and PktCnt = 1.
2. Program the HCCHARn register of the selected channel with ChEna = 1 and the Device's Endpoint characteristics, such as type, speed, and direction.

The application must treat a zero-length data packet as a separate transfer, and cannot combine it with a non-zero length transfer.

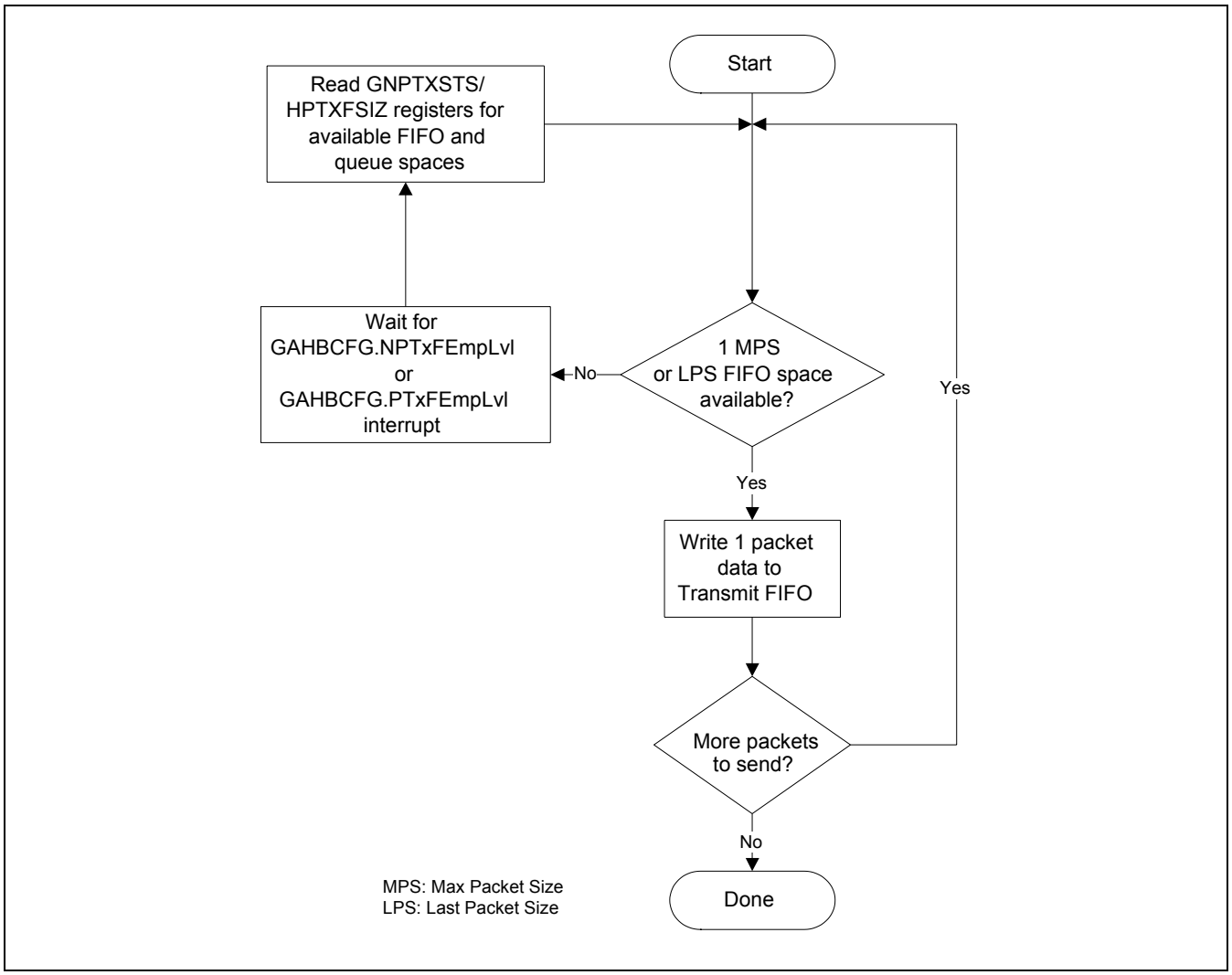
38.9.5 Operational Model

The application must initialize a channel as described in *Channel Initialization* on page 1397 before communicating to the connected Device. This section explains the sequence of operation to be performed for different types of USB transactions.

38.9.5.1 Writing the Transmit FIFO in Slave Mode

The following figure shows the flow diagram for writing to the transmit FIFO in Slave mode. The USB Host automatically writes an entry (OUT request) to the Periodic/Non-periodic Request Queue, along with the last DWORD write of a packet. The application must ensure that at least one free space is available in the Periodic/Non-periodic Request Queue before starting to write to the transmit FIFO. The application must always write to the transmit FIFO in DWORDs. If the packet size is non-DWORD aligned, the application must use padding. The USB Host determines the actual packet size based on the programmed maximum packet size and transfer size.

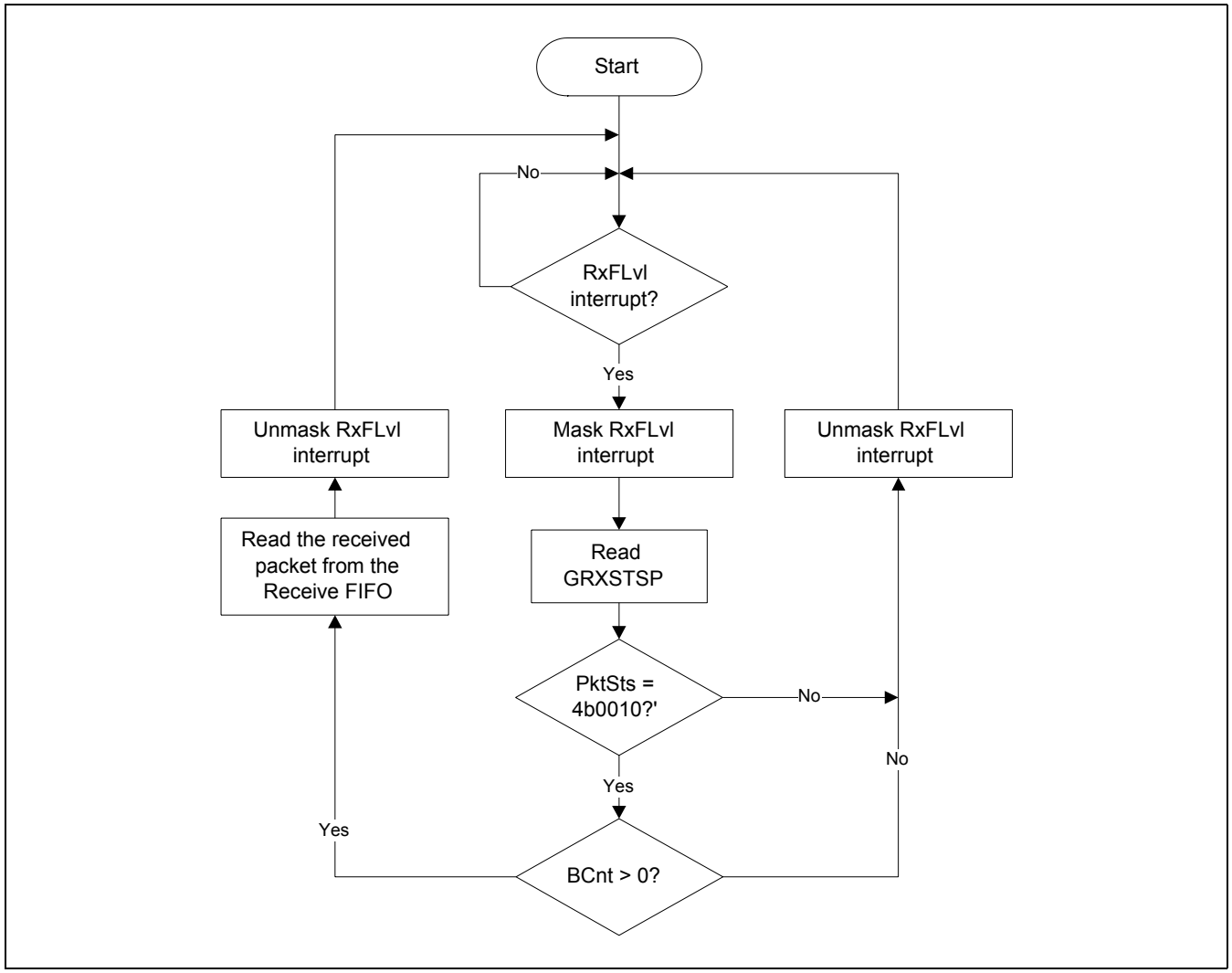
Figure 38-59. Transmit FIFO Write Task in Slave Mode



38.9.5.2 Reading the Receive FIFO in Slave Mode

The following figure shows the flow diagram for reading the receive FIFO in Slave mode. The application must ignore all packet statuses other than IN Data Packet (4'b0010).

Figure 38-60. Receive FIFO Read Task in Slave Mode



38.9.5.3 Bulk and Control OUT/SETUP Transactions in Slave Mode

A typical bulk or control OUT/SETUP pipelined transaction-level operation in Slave mode is shown in *Figure 38-61* on page 1403. See channel 1 (ch_1). Two bulk OUT packets are transmitted. A control SETUP transaction operates the same way but has only one packet. The assumptions are:

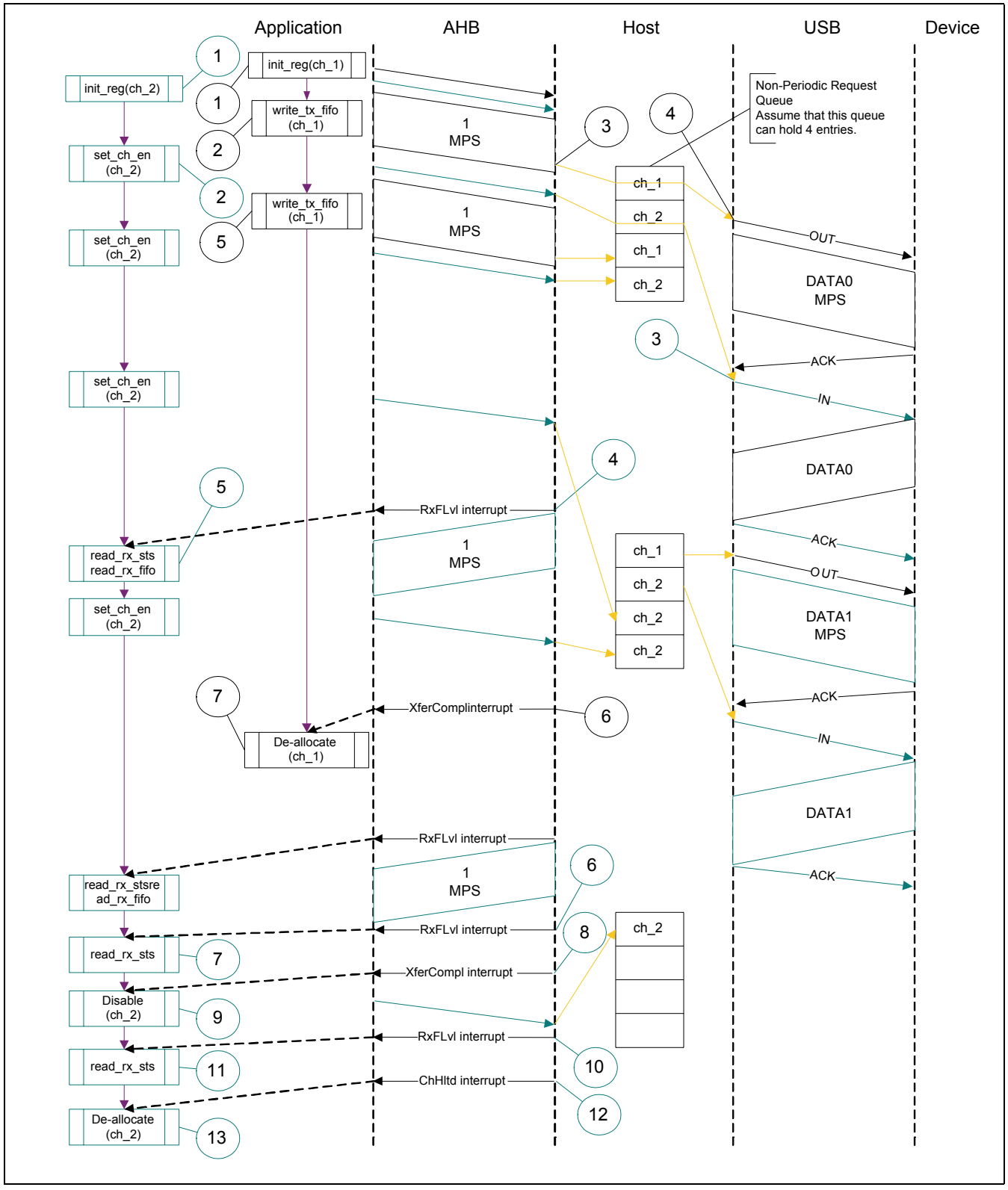
- The application is attempting to send two maximum-packet-size packets (transfer size = 1024B).
- The Non-periodic Transmit FIFO can hold two packets (1KB for HS or 128KB for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control OUT/SETUP Operations

The sequence of operations in *Figure 38-61* (channel 1) is as follows:

1. Initialize channel 1 as explained in *Channel Initialization* on page 1397.
2. Write the first packet for channel 1.
3. Along with the last DWORD write, the function writes an entry to the Non-periodic Request Queue.
4. As soon as the Non-periodic Queue becomes non-empty, the function attempts to send an OUT token in the current frame/microframe.
5. Write the second (last) packet for channel 1.
6. The function generates the XferCompl interrupt as soon as the last transaction is completed successfully.
7. In response to the XferCompl interrupt, de-allocate the channel for other transfers.

Figure 38-61. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in Slave Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control OUT/SETUP transactions in Slave mode is shown in the following table.

Figure 38-62. ISR for Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in Slave Mode

Bulk/Control OUT/SETUP (Non-Split)

```
Unmask (NAK/XactErr/NYET/STALL/XferCompl)
if (XferCompl)
{
Reset Error Count
Mask ACK
De-allocate Channel
}
else if (STALL)
{
Transfer Done = 1
Unmask ChHltd
Disable Channel
}
else if (NAK or XactErr or NYET)
{
Rewind Buffer Pointers
Unmask ChHltd
Disable Channel
   if (XactErr)
   {
   Increment Error Count
   Unmask ACK
   }
   else
   {
   Reset Error Count
   }
}
else if (ChHltd)
{
Mask ChHltd
   if (Transfer Done or (Error_count == 3))
   {
   De-allocate Channel
   }
   else
   {
   Re-initialize Channel (Do ping protocol for HS)
   }
}
else if (ACK)
{
Reset Error Count
Mask ACK
}
```

1. The application is expected to write the data packets into the transmit FIFO as and when the space is available in the transmit FIFO and the Request queue. The application can make use of GINTSTS.NPTxFEmp interrupt to find the transmit FIFO space.

Bulk/Control IN (Non-Split)

```
Unmask (XactErr/XferCompl/BblErr/STALL/DataTglErr)
if (XferCompl)
{
Reset Error Count
Unmask ChHltd
Disable Channel
Reset Error Count
Mask ACK
}
else if (XactErr or BblErr or STALL)
{
Unmask ChHltd
Disable Channel
   if (XactErr)
   {
   Increment Error Count
   Unmask ACK
   }
}
else if (ChHltd)
{
Mask ChHltd
   if (Transfer Done or (Error_count == 3))
   {
   De-allocate Channel
   }
   else
   {
   Re-initialize Channel
   }
}
else if (ACK)
{
Reset Error Count

Mask ACK
}
else if (DataTglErr)
{
 Reset Error Count
}
```

1. The application is expected to write the requests as and when the Request queue space is available and until the XferCompl interrupt is received.
2. The application must clear and never modify the DoPing bit after enabling the channel and until the XferCompl or ChHltd interrupt is received. The function uses the DoPing bit to flush the excessive IN requests after receiving the last or short packet.

38.9.5.4 Bulk and Control IN Transactions in Slave Mode

A typical bulk or control IN pipelined transaction-level operation in Slave mode is shown in *Figure 38-61* on page 1403 See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive two maximum-packet-size packets (transfer size = 1024B).
- The receive FIFO can contain at least one maximum-packet-size packet and two status DWORDs per packet (520B for HS or 72B for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control IN Operations

The sequence of operations in *Figure 38-61* on page 1403 (channel 2) is as follows:

1. Initialize channel 2 as explained in *Channel Initialization* on page 1397.
2. Set the HCCHAR2.ChEna bit to write an IN request to the Non-periodic Request Queue.
3. The function attempts to send an IN token after completing the current OUT transaction.
4. The function generates an RxFLvl interrupt as soon as the received packet is written to the receive FIFO.
5. In response to the RxFLvl interrupt, mask the RxFLvl interrupt and read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. Following this, unmask the RxFLvl interrupt.
6. The function generates the RxFLvl interrupt for the transfer completion status entry in the receive FIFO.
7. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (GRXSTSR.PktSts!= 4'b0010).
8. The function generates the XferCompl interrupt as soon as the receive packet status is read.
9. In response to the XferCompl interrupt, disable the channel (as explained in *Halting a Channel* on page 1398) and stop writing the HCCHAR2 register for further requests. The function writes a channel disable request to the Non-periodic Request Queue as soon as the HCCHAR2 register is written.
10. The function generates the RxFLvl interrupt as soon as the halt status is written to the receive FIFO.
11. Read and ignore the receive packet status.
12. The function generates a ChHltd interrupt as soon the halt status is popped from the receive FIFO.
13. In response to the ChHltd interrupt, de-allocate the channel for other transfers.

Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control IN transactions in Slave mode is shown in *Table 38-62* on page 1404.

38.9.5.5 Bulk and Control OUT/SETUP Transactions in DMA Mode

A typical bulk or control OUT/SETUP operation in DMA mode is shown in *Figure 38-63* on page 1407. See channel 1 (ch_1). Two bulk OUT packets are transmitted. A control SETUP transaction operates the same way but has only one packet. The assumptions are:

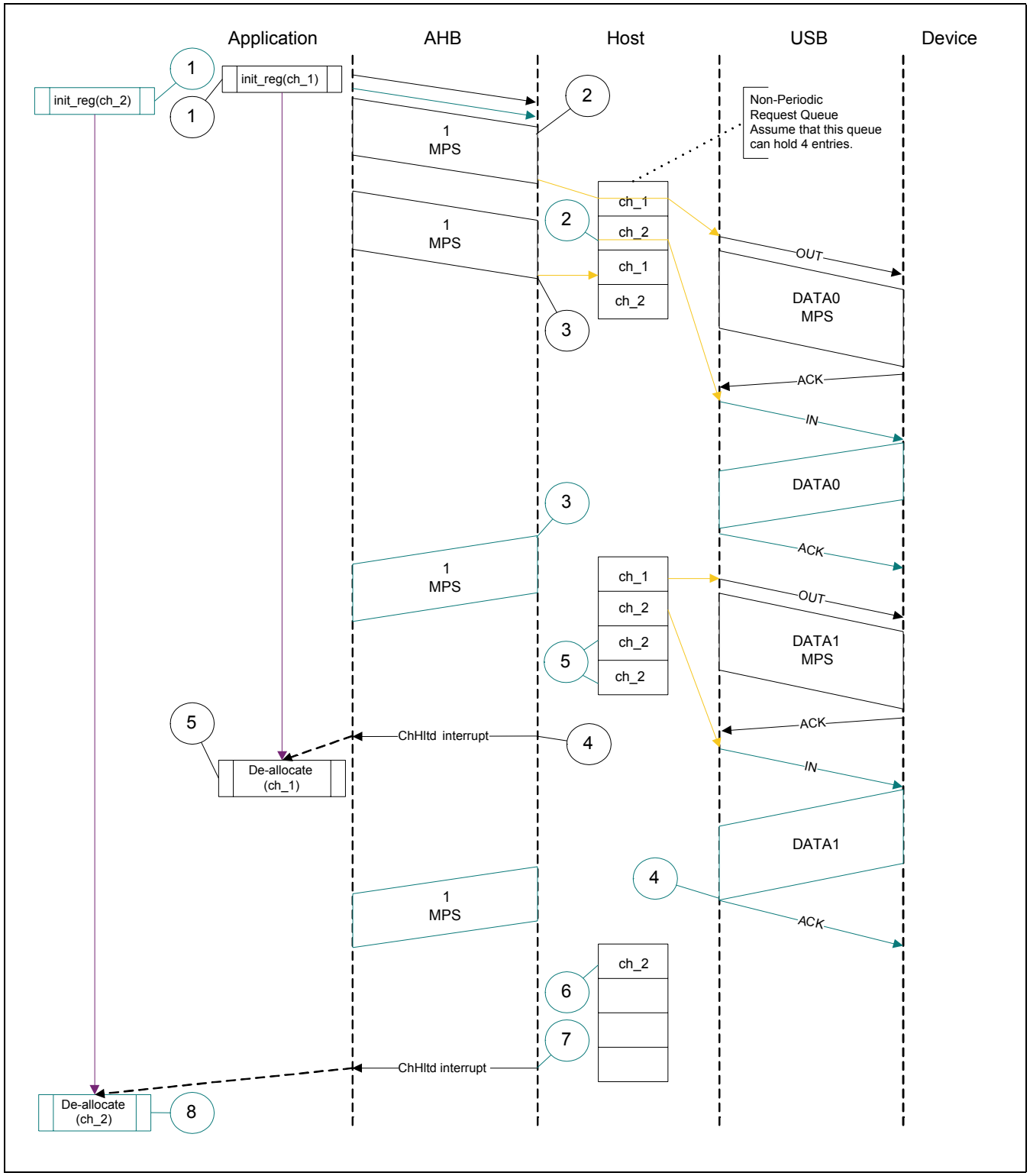
- The application is attempting to send two maximum-packet-size packets (transfer size = 1024B).
- The Non-periodic Transmit FIFO can hold two packets (1 KB for HS or 128B for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control OUT/SETUP Operations

The sequence of operations in *Figure 38-61* on page 1403 (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in *Channel Initialization* on page 1397.
2. The USB Host starts fetching the first packet as soon as the channel is enabled. For internal DMA mode, the USB Host uses the programmed DMA address to fetch the packet.
3. After fetching the last DWORD of the second (last) packet, the USB Host masks channel 1 internally for further arbitration.
4. The USB Host generates a ChHltd interrupt as soon as the last packet is received.
5. In response to the ChHltd interrupt, deallocate the channel for other transfers.

Figure 38-63. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in DMA Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control OUT/ SETUP transactions in DMA mode is shown in the following table.

Figure 38-64. ISRs for Bulk/Control OUT/SETUP and Bulk/Control IN Transactions in DMA Mode

Bulk/Control OUT/SETUP (Non-Split)

```
Unmask (ChHltd)
if (ChHltd)
{
   if (XferCompl or STALL)
   {
   Reset Error Count
   Mask ACK
   De-allocate Channel
   }
   else if (NAK or XactErr or NYET)
   {
   Rewind Buffer Pointers
      if (XactErr)
      {
        if (Error_count == 2)
        {
        De-allocate Channel
        }
        else
        {
        Increment Error Count Re-initialize Channel
        }
      }
      else
      {
      Reset Error Count
      Mask ACK
      }
    Re-initialize Channel
    }
}
else if (ACK)
{
Reset Error Count
Mask ACK
}
```

Bulk/Control IN (Non-Split)

```
Unmask (ChHltd)
if (ChHltd)
{
   if (XferCompl orSTALL or BblErr)
   {
   Reset Error Count
   Mask ACK
   De-allocate Channel
   }
   else if (XactErr)
   {
     if (Error_count == 2)
     {
     De-allocate Channel
     }
     else
     {
     Unmask ACK
     Unmask NAK
     Unmask DataTglErr
     Increment Error Count
     Re-initialize Channel
     }
   }
}
else if (ACK or NAK or DataTglErr)
{
Reset Error Count
Mask ACK
Mask NAK
Mask DataTglErr
}
```

1. As soon as the channel is enabled, the function attempts to fetch and write data packets, in multiples of the maximum packet size, to the transmit FIFO when space is available in the transmit FIFO and the Request queue. The function stops fetching as soon as the last packet is fetched.
2. While continuing the transfer to a high-speed device, the application must set the DoPing bit before enabling the channel if the previous transaction ended with NAK, NYET, or XacrErr response. In this case, the function starts with the ping protocol, then automatically switches to Data Transfer mode.

1. The application must clear and never modify the DoPing bit after enabling the channel and until the ChHltd interrupt is received. The function uses the DoPing bit to flush the excessive IN requests after receiving the last or short packet.

38.9.5.6 Bulk and Control IN Transactions in DMA Mode

A typical bulk or control IN operation in DMA mode is shown in *Figure 38-63* on page 1407. See channel 2 (ch_2).

The assumptions are:

- The application is attempting to receive two maximum-packet-size packets (transfer size = 1024B).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (520B for HS or 72B for FS).
- The Non-periodic Request Queue depth = 4.

Normal Bulk and Control IN Operations

The sequence of operations in *Figure 38-63* on page 1407 (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in *Channel Initialization* on page 1397.
2. The USB Host writes an IN request to the Request queue as soon as channel 2 receives the grant from the arbiter. Arbitration is performed in a round-robin fashion, with fairness.
3. The USB Host starts writing the received data to the system memory as soon as the last byte is received with no errors.
4. When the last packet is received, the USB Host sets an internal flag to remove any extra IN requests from the Request queue.
5. The USB Host flushes the extra requests.
6. The final request to disable channel 2 is written to the Request queue. At this point, channel 2 is internally masked for further arbitration.
7. The USB Host generates the ChHltd interrupt as soon as the disable request comes to the top of the queue.
8. In response to the ChHltd interrupt, deallocate the channel for other transfers.

Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control IN transactions in DMA mode is shown in *Figure 38-64* on page 1408.

38.9.5.7 Control Transactions in Slave Mode

Setup, Data, and Status stages of a control transfer must be performed as three separate transfers. Setup- Data- or Status-stage OUT transactions are performed similarly to the bulk OUT transactions explained in *Bulk and Control OUT/SETUP Transactions in Slave Mode* on page 1401. Data- or Status-stage IN transactions are performed similarly to the bulk IN transactions explained in *Bulk and Control IN Transactions in Slave Mode* on page 1405. For all three stages, the application is expected to set the HCCHAR1.EPType field to Control. During the Setup stage, the application is expected to set the HCTSIZ1.PID field to SETUP.

38.9.5.8 Control Transactions in DMA Mode

Setup, Data, and Status stages of a control transfer must be performed as three separate transfers. Setup- and Data- or Status-stage OUT transactions are performed similarly to the bulk OUT transactions explained in *Bulk and Control OUT/SETUP Transactions in DMA Mode* on page 1405. Data- or Status-stage IN transactions are performed similarly to the bulk IN transactions explained in *Bulk and Control IN Transactions in DMA Mode* on page 1409. For all three stages, the application is expected to set the HCCHAR1.EPType field to Control. During the Setup stage, the application is expected to set the HCTSIZ1.PID field to SETUP.

38.9.5.9 Interrupt OUT Transactions in Slave Mode

A typical interrupt OUT operation in Slave mode is shown in *Figure 38-65* on page 1411. See channel 1 (ch_1). The assumptions are:

- The application is attempting to send one packet in every frame/microframe (up to 1 maximum packet size), starting with the odd frame/microframe (transfer size = 1024B).
- The Periodic Transmit FIFO can hold one packet (1 KB for HS or FS).
- Periodic Request Queue depth = 4.

Normal Interrupt OUT Operation

The sequence of operations in *Figure 38-65* on page 1411 (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR1.OddFrm bit.
2. Write the first packet for channel 1. For a high-bandwidth interrupt transfer, the application must write the subsequent packets up to MC (maximum number of packets to be transmitted in the next frame/microframe times before switching to another channel.
3. Along with the last DWORD write of each packet, the USB Host writes an entry to the Periodic Request Queue.
4. The USB Host attempts to send an OUT token in the next (odd) frame/microframe.
5. The USB Host generates an XferCompl interrupt as soon as the last packet is transmitted successfully.
6. In response to the XferCompl interrupt, re initialize the channel for the next transfer.

Handling Non-ACK Responses

Table 38-66 on page 1412 shows the channel-specific interrupt service routine (ISR) for an interrupt OUT transaction in Slave mode.

Figure 38-65. Normal Interrupt OUT/IN Transactions in Slave Mode

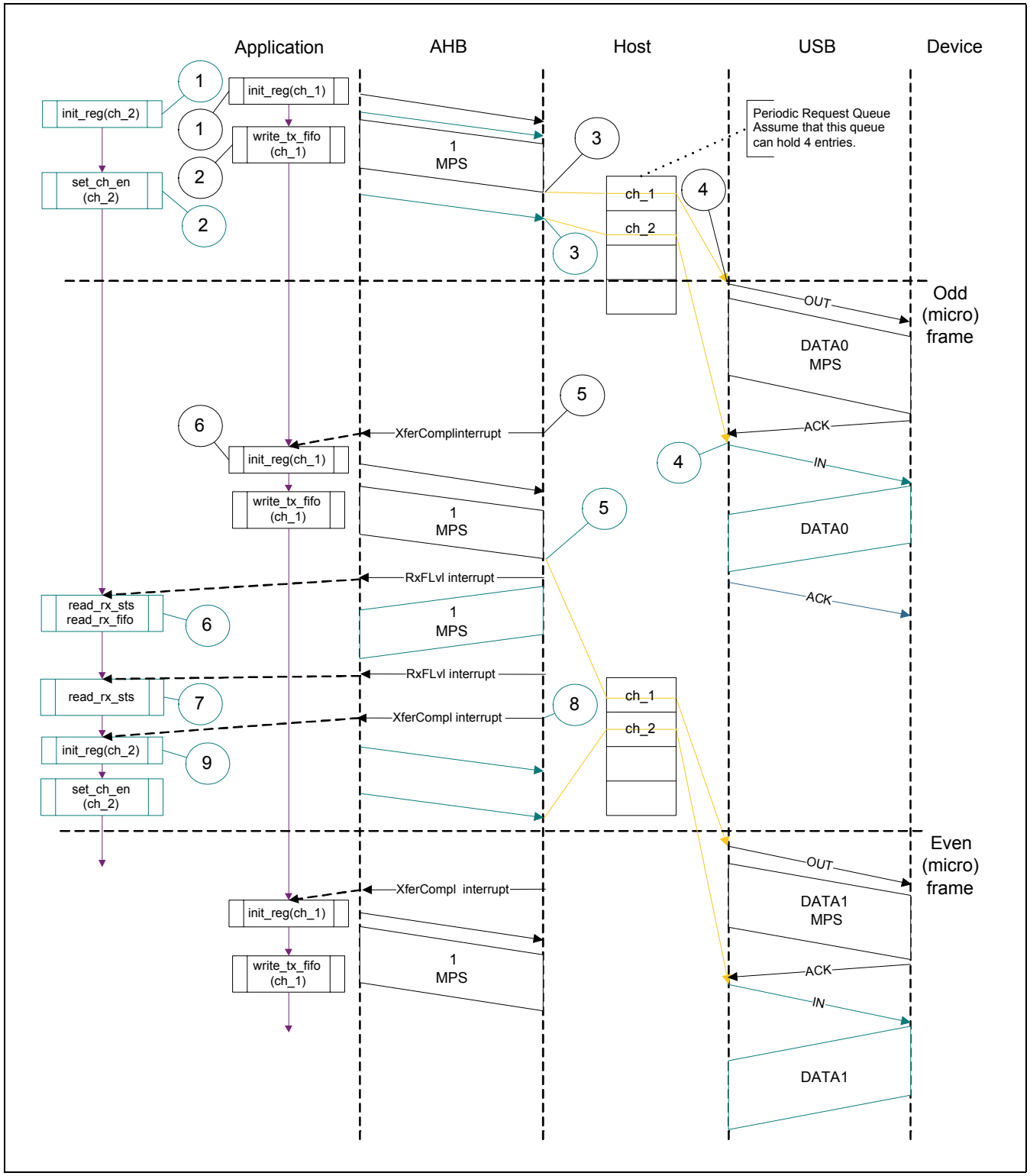


Figure 38-66. ISR for Interrupt OUT/IN Transactions in Slave Mode

Interrupt OUT (Non-Split)

```
Unmask (NAK/XactErr/STALL/XferCompl/FrmOvrun)
if (XferCompl)
{
Reset Error Count
Mask ACK
De-allocate Channel
}
else if (STALL or FrmOvrun)
{
Mask ACK
Unmask ChHltd
Disable Channel
    if (STALL)
    {
    Transfer Done = 1
    }
}
else if (NAKor XactErr)
{
Rewind Buffer Pointers
Reset Error Count
Mask ACK
Unmask ChHltd
Disable Channel
}
else if (ChHltd)
{
Mask ChHltd
    if (Transfer Done or (Error_count == 3))
    {
    De-allocate Channel
    }
    else
    {
    Re-initialize Channel (in next b_interval -1 uF/F)
    }
}
else if (ACK)
{
Reset Error Count
Mask ACK
}
```

1. The application is expected to write the data packets into the transmit FIFO when the space is available in the transmit FIFO and the Request queue up to the count specified in the MC field before switching to another channel. The application uses the GINTSTS.NPTxFEmp interrupt to find the transmit FIFO space.

Interrupt IN (Non-Split)

```
Unmask (NAK/XactErr/XferCompl/BblErr/STALL/FrmOvrun/DataTglErr)
if (XferCompl)
{
Reset Error Count
Mask ACK
    if (HCTSIZn.PktCnt == 0)
    {
    e-allocate Channel
    }
    else
    {
    Transfer Done = 1
    Unmask ChHltd
    Disable Channel
    }
}
else if (STALL or FrmOvrun or NAK or DataTglErr or BblErr)
{
Mask ACK
Unmask ChHltd
Disable Channel
    if (STALL or BblErr)
    {
    Reset Error Count
    Transfer Done = 1
    }
    else if (!FrmOvrun)
    {
    Reset Error Count
    }
}
else if (XactErr)
{
Increment Error Count
Unmask ACK
Unmask ChHltd
Disable Channel
}
else if (ChHltd)
{
Mask ChHltd
    if (Transfer Done or (Error_count == 3))
    {
    De-allocate Channel
    }
    else
    {
    Re-initialize Channel (in next b_interval - 1 uF/F)
    }
}
else if (ACK)
{
Reset Error Count
Mask ACK
}
```

1. The application is expected to write the requests for the same channel when the Request queue space is available up to the count specified in the MC field before switching to another channel (if any).

38.9.5.10 Interrupt IN Transactions in Slave Mode

A typical interrupt IN operation in Slave mode is shown in *Figure 38-65* on page 1411. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame/microframe, starting with odd. (transfer size = 1024B).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1032B for HS or 1031B for FS).
- Periodic Request Queue depth = 4.

Normal Interrupt IN Operation

The sequence of operations in *Figure 38-65* on page 1411 (channel 2) is as follows:

1. Initialize channel 2 as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR2.OddFrm bit.
2. Set the HCCHAR2.ChEna bit to write an IN request to the Periodic Request Queue. For a high-bandwidth interrupt transfer, the application must write the HCCHAR2 register MC (maximum number of expected packets in the next frame/microframe) times before switching to another channel.
3. The USB Host writes an IN request to the Periodic Request Queue for each HCCHAR2 register write with a ChEna bit set.
4. The USB Host attempts to send an IN token in the next (odd) frame/microframe.
5. As soon as the IN packet is received and written to the receive FIFO, the USB Host generates an RxFLvl interrupt.
6. In response to the RxFLvl interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RxFLvl interrupt before reading the receive FIFO, and unmask after reading the entire packet.
7. The function generates the RxFLvl interrupt for the transfer completion status entry in the receive FIFO. The application must read and ignore the receive packet status when the receive packet status is not an IN data packet (GRXSTSR.PktSts != 4'b0010).
8. The function generates an XferCompl interrupt as soon as the receive packet status is read.
9. In response to the XferCompl interrupt, read the HCTSIZ2.PktCnt field. If HCTSIZ2.PktCnt != 0, disable the channel (as explained in *Halting a Channel* on page 1398) before re-initializing the channel for the next transfer, if any). If HCTSIZ2.PktCnt == 0, re initialize the channel for the next transfer. This time, the application must reset the HCCHAR2.OddFrm bit.

Handling Non-ACK Responses

The channel-specific interrupt service routine for an interrupt IN transaction in Slave mode is shown in *Table 38-66* on page 1412.

38.9.5.11 Interrupt OUT Transactions in DMA Mode

A typical interrupt OUT operation in DMA mode is shown in *Figure 38-67* on page 1415. See channel 1 (ch_1). The assumptions are:

- The application is attempting to transmit one packet in every frame/microframe (up to 1 maximum packet size of 1024B).
- The Periodic Transmit FIFO can hold one packet (1 KB for HS/FS).
- Periodic Request Queue depth = 4.

Normal Interrupt OUT Operation

1. Initialize and enable channel 1 as explained in *Channel Initialization* on page 1397.
2. The USB Host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last DWORD fetch. In high-bandwidth transfers, the USB Host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The USB Host attempts to send the OUT token in the beginning of the next odd frame/microframe.
4. After successfully transmitting the packet, the USB Host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinstalled the channel for the next transfer.

Handling Non-ACK Responses

Figure 38-68 on page 1416 shows the channel-specific ISR for an interrupt OUT transaction in DMA mode.

Figure 38-67. Normal Interrupt OUT/IN Transactions in DMA Mode

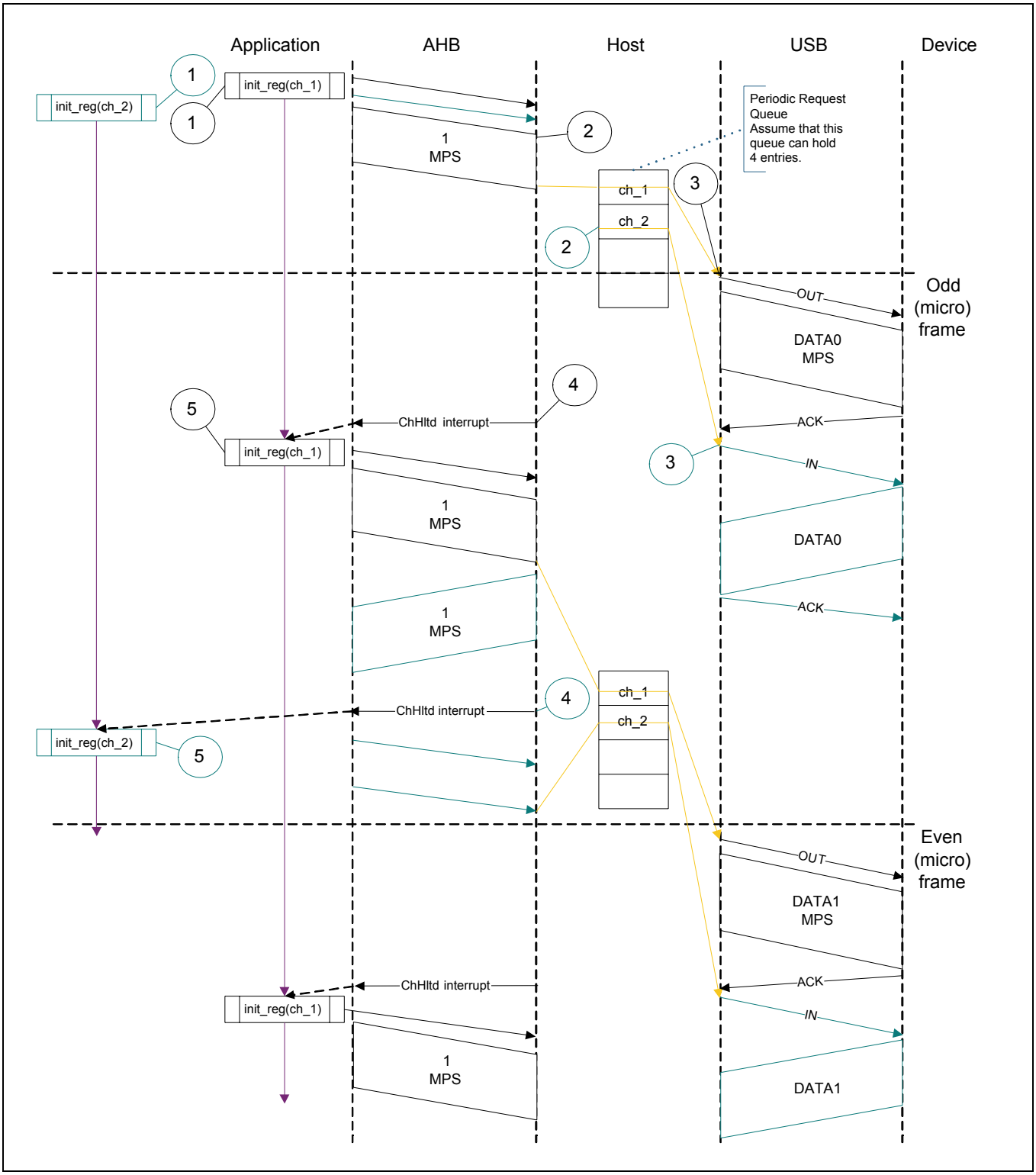


Figure 38-68. ISR for Interrupt OUT/IN Transactions in DMA Mode

Interrupt OUT (Non-Split)

```
Unmask (ChHltd)
if (ChHltd)
{
   if (XferCompl)
   {
   Reset Error Count
   Mask ACK
      if (Transfer Done)
      {
      De-allocate Channel
      }
      else
      {
      Re-initialize Channel (in next b_interval - 1 uF/F)
      }
   }
   else if (STALL)
   {
   Transfer Done = 1
   Reset Error Count
   Mask ACK
   De-allocate Channel
   }
   else if (NAK or FrmOvrun)
   {
   Mask ACK
   Rewind Buffer Pointers
   Re-initialize Channel (in next b_interval -1 uF/F)
      if (NAK)
      {
      Reset Error Count
      }
   }
   else if (XactErr)
   {
      if (Error_count == 2)
      {
      De-allocate Channel
      }
      else
      {
      Increment Error Count
      Rewind Buffer Pointers
      Unmask ACK
      Re-initialize Channel (in next b_interval - 1 uF/F)
      }
   }
}
else if (ACK)
{
Reset Error Count
Mask ACK
}
```

1. As soon as the channel is enabled, the function attempts to fetch and write data packets, in maximum packet size multiples, to the transmit FIFO when the space is available in the transmit FIFO and the Request queue. The function stops fetching as soon as the last packet is fetched (the number of packets is determined by the MC field of the HCCHARn register).

Interrupt IN (Non-Split)

```
Unmask (ChHltd)
if (ChHltd)
{
    if (XferCompl)
   {
   Reset Error Count
   Mask ACK
         if (Transfer Done)
         {
         De-allocate Channel
         }
         else
         {
         Re-initialize Channel (in next b_interval - 1 uF/F)
         }
      }
      else if (STALL or BblErr)
      {
      Reset Error Count
      Mask ACK
      De-allocate Channel
      }
      else if (NAK or DataTglErr or FrmOvrun)
      {
      Mask ACK
      Re-initialize Channel (in next b_interval - 1 uF/F)
         if (DataTglErr or NAK)
         {
         Reset Error Count
         }
      }
      else if (XactErr)
      {
      if (Error_count == 2)
      {
      De-allocate Channel
      }
      else
      {
      Increment Error Count
      Unmask ACK
      Re-initialize Channel (in next b_interval - 1 uF/F)
      }
   }
}
else if (ACK)
{
Reset Error Count
Mask ACK
```

1. As soon as the channel is enabled, the function attempts to write the requests into the Request queue when the space is available up to the count specified in the MC field.

38.9.5.12 Interrupt IN Transactions in DMA Mode

A typical isochronous IN operation in DMA mode is shown in *Figure 38-67* on page 1415. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet in every frame/microframe (up to 1 maximum packet size of 1024B).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1032B for HS/FS).
- Periodic Request Queue depth = 4.

Normal Interrupt IN Operation

The sequence of operations in *Figure 38-67* on page 1415 (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in *Channel Initialization* on page 1397.
2. The USB Host writes an IN request to the Request queue as soon as the channel 2 gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the USB Host writes consecutive writes up to MC times.
3. The USB Host attempts to send an IN token at the beginning of the next (odd) frame/microframe.
4. As soon the packet is received and written to the receive FIFO, the USB Host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinstalled the channel for the next transfer.

Handling Non-ACK Responses

The channel-specific interrupt service routine for Interrupt IN transaction in DMA mode is shown in *Figure 38-68* on page 1416.

38.9.5.13 Isochronous OUT Transactions in Slave Mode

A typical isochronous OUT operation in Slave mode is shown in *Figure 38-69* on page 1418. See channel 1 (ch_1). The assumptions are:

- The application is attempting to send one packet every frame/microframe (up to 1 maximum packet size), starting with an odd frame/microframe. (transfer size = 1024B).
- The Periodic Transmit FIFO can hold one packet (1 KB for HS/FS).
- Periodic Request Queue depth = 4.

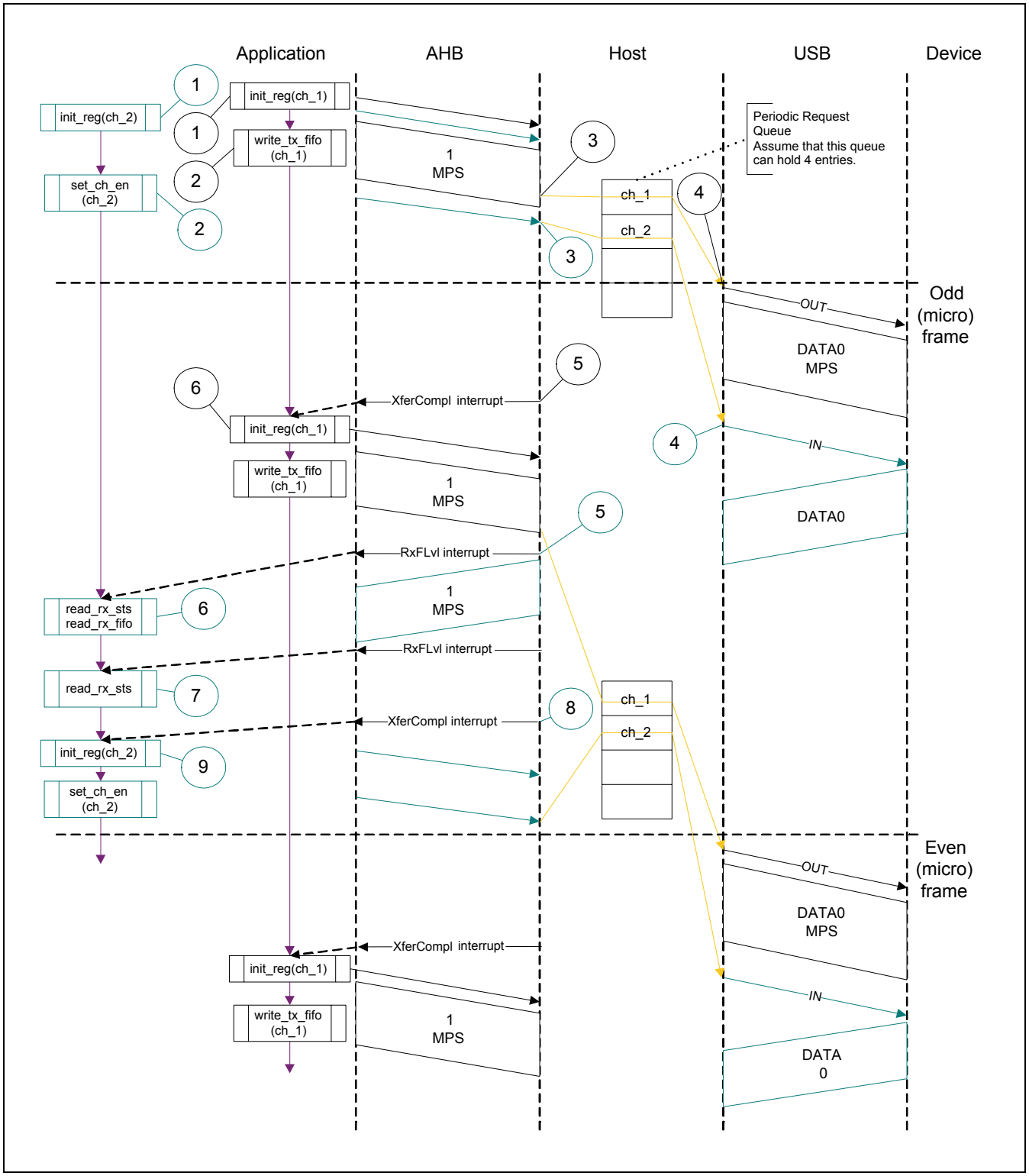
Normal Isochronous OUT Operation

The sequence of operations in *Figure 38-69* on page 1418 (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR1.OddFrm bit.
2. Write the first packet for channel 1. For a high-bandwidth isochronous transfer, the application must write the subsequent packets up to MC (maximum number of packets to be transmitted in the next frame/microframe) times before switching to another channel.
3. Along with the last DWORD write of each packet, the USB Host writes an entry to the Periodic Request Queue.
4. The USB Host attempts to send the OUT token in the next frame/microframe (odd).
5. The USB Host generates the XferCompl interrupt as soon as the last packet is transmitted successfully.

6. In response to the XferCompl interrupt, reinstalled the channel for the next transfer.

Figure 38-69. Normal Isochronous OUT/IN Transactions in Slave Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for an isochronous OUT transaction in Slave mode is shown in the following table.

Figure 38-70. ISR for Isochronous OUT/IN Transactions in Slave Mode

Isochronous OUT (Non-Split)

```
Unmask (FrmOvrun/XferCompl)
if (XferCompl)
{
De-allocate Channel
}
else if (FrmOvrun)
{
Unmask ChHltd
Disable Channel
}
else if (ChHltd)
{
Mask ChHltd
De-allocate Channel
}
```

Isochronous IN (Non-Split)

```
Unmask (XactErr/XferCompl/FrmOvrun/BblErr)
if (XferCompl or FrmOvrun)
{
    if (XferCompl and (HCTSIZn.PktCnt == 0))
    {
    Reset Error Count De-allocate Channel
    }
    else
    {
    Unmask ChHltd
    Disable Channel
    }
}
else if (XactError BblErr)
{
Increment Error Count
UnmaskChHltd
Disable Channel
}
else if (ChHltd)
{
Mask ChHltd
    if (Transfer Done or (Error_count == 3))
    {
    De-allocate Channel
    }
    else
    {
    Re-initialize Channel
    }
}
```

38.9.5.14 Isochronous IN Transactions in Slave Mode

A typical isochronous IN operation in Slave mode is shown in *Figure 38-69* on page 1418. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet (up to 1 maximum packet size) in every frame/microframe starting with the next odd frame/microframe. (transfer size = 1024B).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet (1032B for HS or 1031B for FS).
- Periodic Request Queue depth = 4.

Normal Isochronous IN Operation

The sequence of operations in *Figure 38-69* on page 1418 (channel 2) is as follows:

1. Initialize channel 2 as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR2.OddFrm bit.

2. Set the HCCHAR2.ChEna bit to write an IN request to the Periodic Request Queue. For a high-bandwidth isochronous transfer, the application must write the HCCHAR2 register MC (maximum number of expected packets in the next frame/microframe) times before switching to another channel.
3. The USB Host writes an IN request to the Periodic Request Queue for each HCCHAR2 register write with the ChEna bit set.
4. The USB Host attempts to send an IN token in the next odd frame/microframe.
5. As soon as the IN packet is received and written to the receive FIFO, the USB Host generates an RxFLvl interrupt.
6. In response to the RxFLvl interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RxFLvl interrupt before reading the receive FIFO, and unmask it after reading the entire packet.
7. The function generates an RxFLvl interrupt for the transfer completion status entry in the receive FIFO. This time, the application must read and ignore the receive packet status when the receive packet status is not an IN data packet (GRXSTSR.PktSts != 4'b0010).
8. The function generates an XferCompl interrupt as soon as the receive packet status is read.
9. In response to the XferCompl interrupt, read the HCTSIZ2.PktCnt field. If HCTSIZ2.PktCnt != 0, disable the channel (as explained in *Halting a Channel* on page 1398) before re-initializing the channel for the next transfer, if any. If HCTSIZ2.PktCnt == 0, reinstalled the channel for the next transfer. This time, the application must reset the HCCHAR2.OddFrm bit.

Handling Non-ACK Responses

The channel-specific interrupt service routine for an isochronous IN transaction in Slave mode is shown in *Table 38-70* on page 1419.

38.9.5.15 Isochronous OUT Transactions in DMA Mode

A typical isochronous OUT operation in DMA mode is shown in *Figure 38-71* on page 1421. See channel 1 (ch_1).

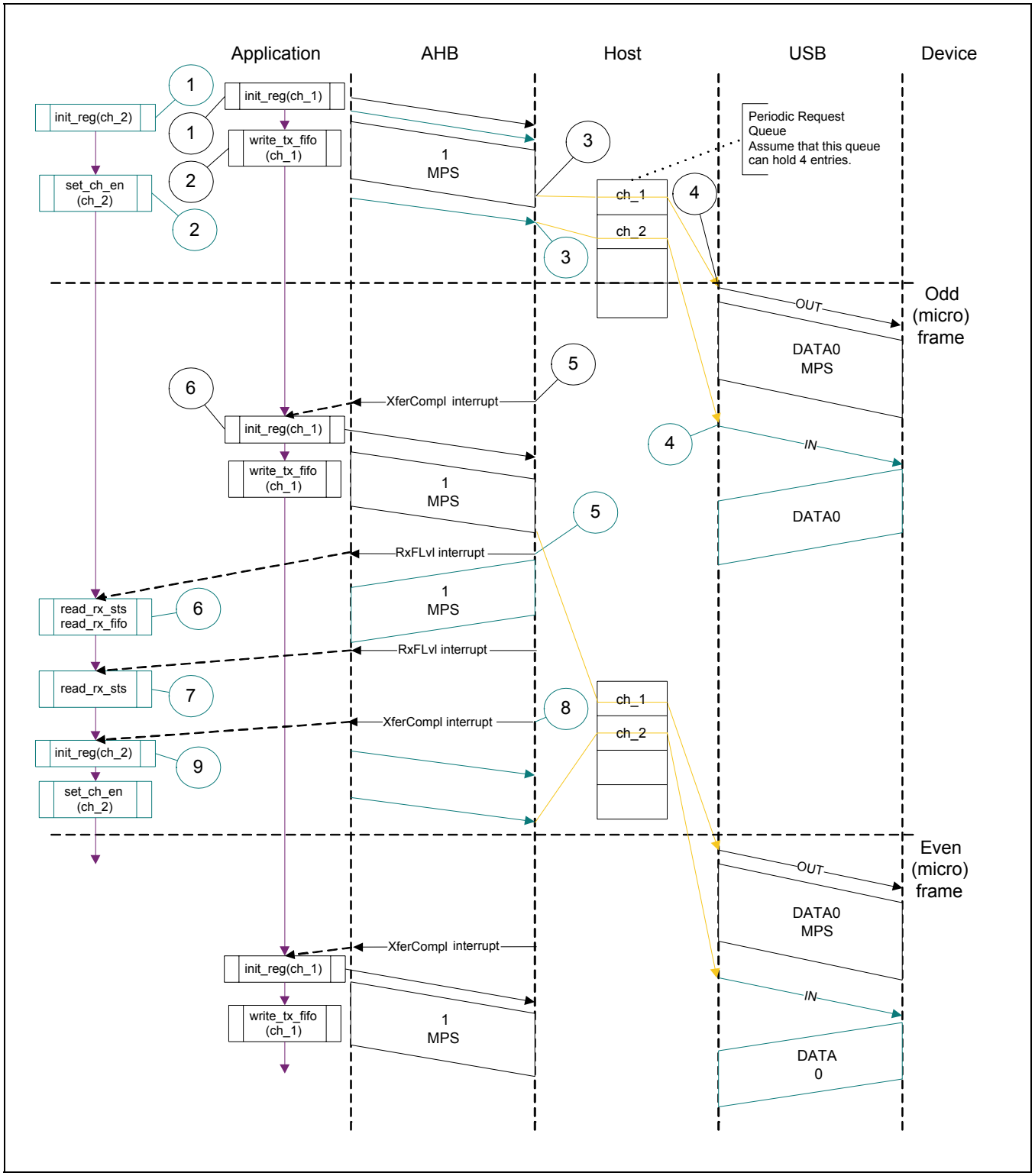
The assumptions are:

- The application is attempting to transmit one packet every frame/microframe (up to 1 maximum packet size of 1024B).
- The Periodic Transmit FIFO can hold one packet (1KB for HS/FS).
- Periodic Request Queue depth = 4.

Normal Isochronous OUT Operation

1. Initialize and enable channel 1 as explained in *Channel Initialization* on page 1397.
2. The USB Host starts fetching the first packet as soon as the channel is enabled, and writes the OUT request along with the last DWORD fetch. In high-bandwidth transfers, the USB Host continues fetching the next packet (up to the value specified in the MC field) before switching to the next channel.
3. The USB Host attempts to send an OUT token in the beginning of the next (odd) frame/microframe.
4. After successfully transmitting the packet, the USB Host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinstalled the channel for the next transfer.

Figure 38-71. Normal Isochronous OUT/IN Transactions in DMA Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for isochronous OUT transaction in DMA mode is shown in the following table.

Figure 38-72. ISR for Isochronous OUT/IN Transactions in DMA Mode

Isochronous OUT (Non-Split)

```
Unmask (ChHltd)
if (ChHltd)
{
   if (XferCompl or FrmOvrun)
   {
   De-allocate Channel
   }
}
```

Isochronous IN (Non-Split)

```
Unmask (ChHltd)
if (ChHltd)
{
   if (XferCompl or FrmOvrun)
   {
      if (XferCompl and (HCTSIZn.PktCnt == 0))
      {
      Reset Error Count
      De-allocate Channel
      }
      else
      {
      De-allocate Channel
      }
   }
   else if (XactErr or BblErr)
   {
      if (Error_count == 2)
      {
      De-allocate Channel
      }
      else
      {
      Increment Error Count
      Re-enable Channel (in next b_interval - 1 uF/F)
      }
   }
}
```

38.9.5.16 Isochronous IN Transactions in DMA Mode

A typical isochronous IN operation in DMA mode is shown in *Figure 38-71* on page 1421. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet in every frame/microframe (up to 1 maximum packet size of 1024B).
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDS per packet (1032B for HS/FS).
- Periodic Request Queue depth = 4.

Normal Isochronous IN Operation

The sequence of operations in *Figure 38-71* on page 1421 (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in *Channel Initialization* on page 1397.
2. The USB Host writes an IN request to the Request queue as soon as the channel 2 gets the grant from the arbiter (round-robin with fairness). In high-bandwidth transfers, the USB Host performs consecutive writes up to MC times.
3. The USB Host attempts to send an IN token at the beginning of the next (odd) frame/microframe.
4. As soon the packet is received and written to the receive FIFO, the USB Host generates a ChHltd interrupt.
5. In response to the ChHltd interrupt, reinstalled the channel for the next transfer.

Handling Non-ACK Responses

The channel-specific interrupt service routine for an isochronous IN transaction in DMA mode is shown in

Figure 38-72 on page 1422.

38.9.5.17 Bulk and Control OUT/SETUP Split Transactions in Slave Mode

A typical bulk or control SETUP/OUT operation in Slave mode is shown in *Figure 38-73* on page 1424. See channel 1 (ch_1). Two bulk OUT packets are transmitted. A control SETUP transaction operates the same way but has only one packet. The assumptions are:

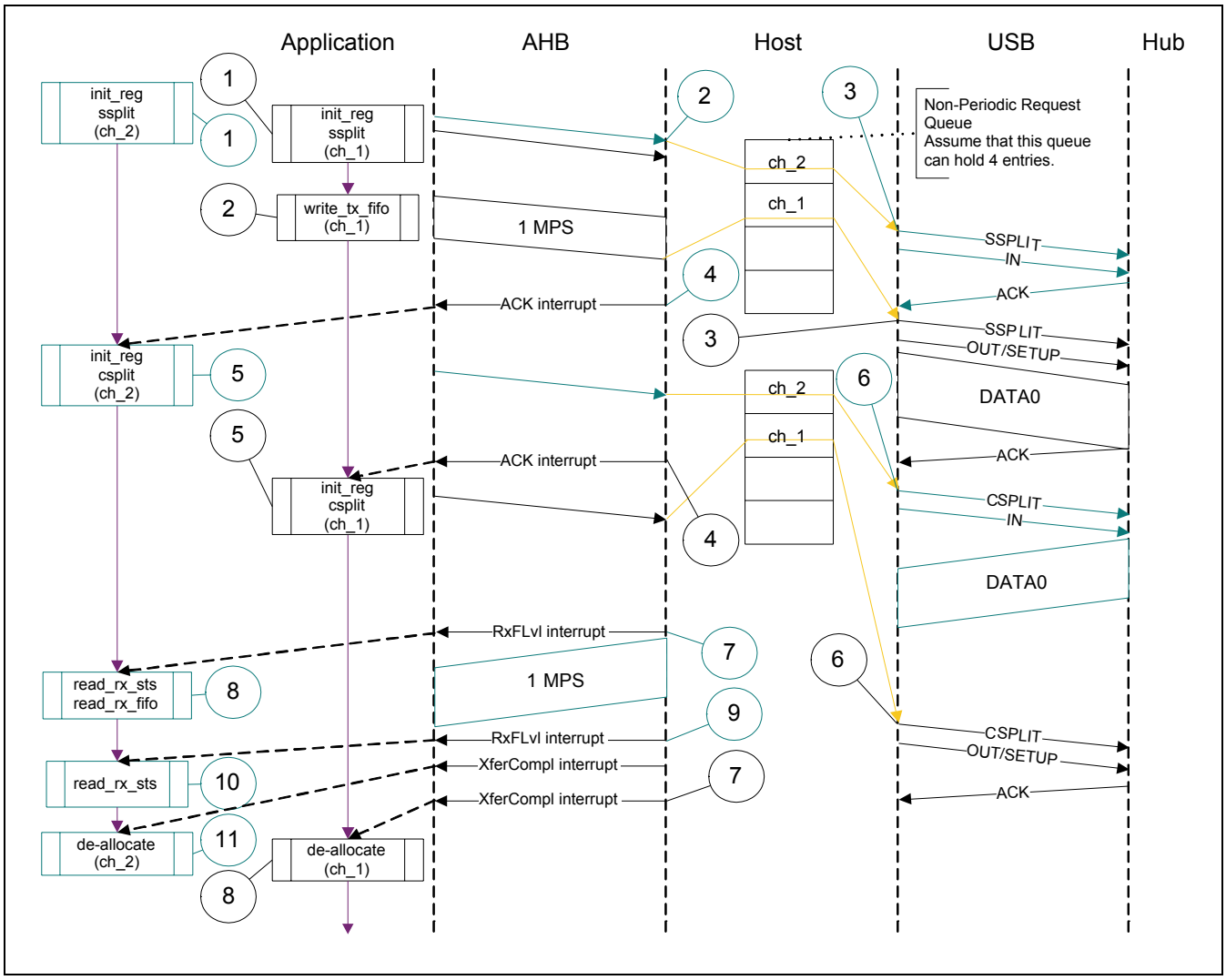
- The application is attempting to transmit one packet.

Normal Bulk and Control OUT/SETUP Split Operations

The sequence of operations in *Figure 38-73* on page 1424 (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in *Channel Initialization* on page 1397.
2. Write the packet for channel 1. Along with the last DWORD write of the packet, the USB Host writes an entry to the Periodic Request Queue.
3. The USB Host sends an OUT token.
4. The USB Host generates an ACK interrupt as soon as the start split transaction completes successfully.
5. In response to the ACK interrupt, set the HCSPLT1.ComplSplt to send the complete split.
6. The OTH Host sends out the complete split transaction.
7. The USB Host generates the XferCompl interrupt after successfully completing the complete split transaction.
8. In response to XferCompl interrupt, deallocate the channel.

Figure 38-73. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Split Transactions in Slave Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control OUT/SETUP split transactions in Slave mode is shown in the following table.

Figure 38-74. ISR for Bulk/Control OUT/SETUP and Bulk/Control IN Split Transactions in Slave Mode

	Bulk/Control OUT/SETUP (Split)	**Bulk/Control IN (Split)**
Start Split	Unmask (ACK/NAK/XactErr) *if (**ACK**)* { Reset Error Count Do Complete Split } *else if (**NAK**)* { Rewind Buffer Pointers Retry Start Split } *else if (**XactErr**)* { Rewind Buffer Pointers Increment Error Count if (Error_count < 3) { Retry Start Split } *else* { Unmask **ChHltd** Disable Channel } } *else if (**ChHltd**)* { Mask **ChHltd** De-allocate Channel }	Unmask (ACK/NAK/XactErr/ DataTglErr) *if (**ACK**)* { Reset Error Count Do Complete Split } *else if (**NAK**)* { Retry Start Split } *else if (**XactErr/DataTglErr**)* { Increment Error Count *if (Error_count < 3)* { Retry Start Split } *else* { Unmask **ChHltd** Disable Channel } } *else if (**ChHltd**)* { Mask **ChHltd** De-allocate Channel }

Figure 38-74. ISR for Bulk/Control OUT/SETUP and Bulk/Control IN Split Transactions in Slave Mode

	Bulk/Control OUT/SETUP (Split)	Bulk/Control IN (Split)
Complete Split	Unmask (NAK/XactErr/NYET/STALL/XferCompl) *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { Rewind Buffer Pointers Retry Start Split } *else if (**NYET**)* { Retry Complete Split } *else if (**STALL**)* { Unmask **ChHltd** Disable Channel } *else if (**XactErr**)* { Rewind Buffer Pointers Increment Error Count if (Error_count < 3) { Retry Start Split } *else* { Unmask **ChHltd** Disable Channel } } *else if (**ChHltd**)* { Mask **ChHltd** De-allocate Channel }	Unmask (NAK/XactErr/NYET/STALL/XferCompl) *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { Retry Start Split } *else if (**NYET**)* { Retry Complete Split } *else if (**STALL** or **BblErr**)* { Unmask **ChHltd** Disable Channel } *else if (**XactErr**)* { Increment Error Count *if (Error_count < 3)* { Retry Start Split } *else* { Unmask ChHltd Disable Channel } } *else if (**ChHltd**)* { Mask ChHltd De-allocate Channel }

38.9.5.18 Bulk and Control IN Split Transactions in Slave Mode

A typical bulk or control IN operation in Slave mode is shown in *Figure 38-73* on page 1424 (see channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one packet.

Normal Bulk and Control IN Split Operations

The sequence of operations in *Figure 38-73* on page 1424 (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in *Channel Initialization* on page 1397.
2. The USB Host writes the Start Split request to the Periodic Request Queue as soon as the HCCHAR2 register is written.
3. The USB Host sends the Start Split IN token.
4. As soon as the IN token is transmitted, the USB Host generates an ACK interrupt.

5. In response to the ACK interrupt, set the HCSPLT2.ComplSplt bit to send the complete split token.
6. The USB Host sends the complete split token.
7. As soon as the received packet is written to the receive FIFO, the USB Host generates the RxFLvl interrupt.
8. In response to the RxFLvl interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RxFLvl interrupt before reading the receive FIFO and unmask it after reading the entire packet.
9. The function generates the RxFLvl interrupt for the transfer completion status entry in the receive FIFO.
10. The application must read the receive packet status and, when the receive packet status is not an IN data packet (GRXSTSR.PktSts != 4'b0010), ignore it.
11. The function generates the XferCompl interrupt as soon as the receive packet status is read. In response to the XferCompl interrupt, read the HCTSIZ2.PktCnt field. If HCTSIZ2.PktCnt != 0, disable the channel (as explained in *Halting a Channel* on page 1398) before re-initializing the channel for the next transfer, if any.

Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control IN split transactions in Slave mode is shown in *Table 38-74* on page 1425.

38.9.5.19 Bulk and Control OUT/SETUP Split Transactions in DMA Mode

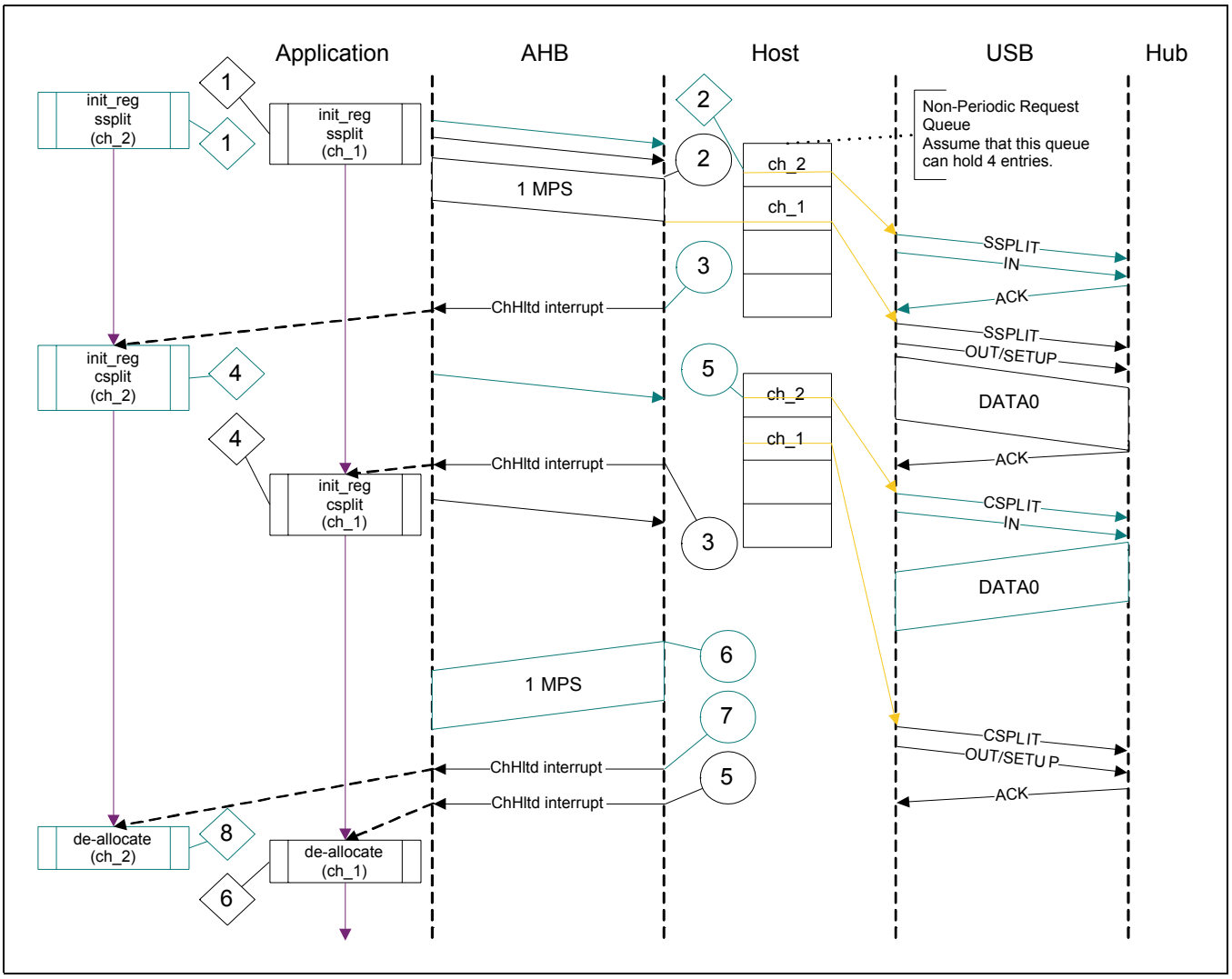
A typical bulk or control OUT/SETUP operation in DMA mode is shown in *Figure 38-75* on page 1428. See channel 1 (ch_1). Two bulk OUT packets are transmitted. A control SETUP transaction operates the same way but has only one packet. It is assumed that the application is attempting to transmit one packet.

Normal Bulk and Control OUT/SETUP Split Operations

The sequence of operations in *Figure 38-75* on page 1428 (channel 1) is as follows:

1. Initialize and enable channel 1 for start split as explained in *Channel Initialization* on page 1397.
2. The USB Host starts fetching the first packet as soon the channel is enabled and writes the OUT request along with the last DWORD fetch.
3. After successfully transmitting start split, the USB Host generates the ChHltd interrupt.
4. In response to the ChHltd interrupt, set the HCSPLT1.ComplSplt bit to send the complete split.
5. After successfully transmitting complete split, the USB Host generates the ChHltd interrupt.
6. In response to the ChHltd interrupt, deallocate the channel.

Figure 38-75. Normal Bulk/Control OUT/SETUP and Bulk/Control IN Split Transactions in DMA Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control OUT/SETUP split transactions in DMA mode is shown in *Figure 38-76* on page 1429.

Figure 38-76. ISR for Bulk/Control OUT/SETUP and Bulk/Control IN Split Transactions in DMA Mode

	Bulk/Control OUT/SETUP (Split)	**Bulk/Control IN (Split)**
Start Split	Unmask (ChHltd) *if (**ChHltd**)* { *if (**ACK**)* { Reset Error Count Do Complete Split } *else if (**NAK**)* { Rewind Buffer Pointers Retry Start Split } *else if (**XactErr**)* { Rewind Buffer Pointers Increment Error Count if (error_count < 3) { Retry Start Split } *else* { De-allocate Channel } } }	Unmask (ChHltd) *if (**ChHltd**)* { *if (**ACK**)* { Reset Error Count Do Complete Split } *else if (**NAK**)* { Retry Start Split } *else if (**XactErr**)* { Increment Error Count *if (error_count < 3)* { Retry Start Split } *else* { De-allocate Channel } }
Complete Split	Unmask (**ChHltd**) *if (**ChHltd**)* { *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { Rewind Buffer Pointers Retry Start Split } *else if (**NYET**)* { Retry Complete Split } *else if (**STALL**)* { De-allocate Channel } *else if (**XactErr**)* { Rewind Buffer Pointers Increment Error Count *if (error_count < 3)* { Retry Start Split } *else* { De-allocate Channel } } }	Unmask (ChHltd) *if (**ChHltd**)* { *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { Retry Start Split } *else if (**NYET**)* { Retry Complete Split } *else if (**STALL/BblErr**)* { De-allocate Channel } *else if (**XactErr**)* { Increment Error Count *if (error_count < 3)* { Retry Start Split } *else* { De-allocate Channel } } }

38.9.5.20 Bulk/Control IN Split Transactions in DMA Mode

A typical bulk or control IN operation in DMA mode is shown in *Figure 38-75* on page 1428. See channel 1 (ch_1).

The assumptions are:

- The application is attempting to receive one packet.

Normal Bulk and Control IN Split Operations

The sequence of operations in *Figure 38-75* on page 1428 (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in *Channel Initialization* on page 1397.
2. The USB Host writes the start split request to the non-periodic request after getting the grant from the arbiter. The USB Host masks the channel 2 internally for the arbitration after writing the request.
3. As soon as the IN token is transmitted, the USB Host generates the ChHltd interrupt.
4. In response to the ChHltd interrupt, set the HCSPLT2.ComplSplt bit to send the complete split token. This unmasks channel 2 for arbitration.
5. The USB Host writes the complete split request to the non-periodic request after receiving the grant from the arbiter.
6. The USB Host starts writing the packet to the system memory after receiving the packet successfully.
7. As soon as the received packet is written to the system memory, the USB Host generates a ChHltd interrupt.
8. In response to the ChHltd interrupt, deallocate the channel.

Handling Non-ACK Responses

The channel-specific interrupt service routine for bulk and control IN split transactions in DMA mode is shown in *Figure 38-76* on page 1429.

38.9.5.21 Interrupt OUT Split Transactions in Slave Mode

A typical interrupt OUT split operation in Slave mode is shown in *Figure 38-77* on page 1431. See channel 1 (ch_1). The assumptions are:

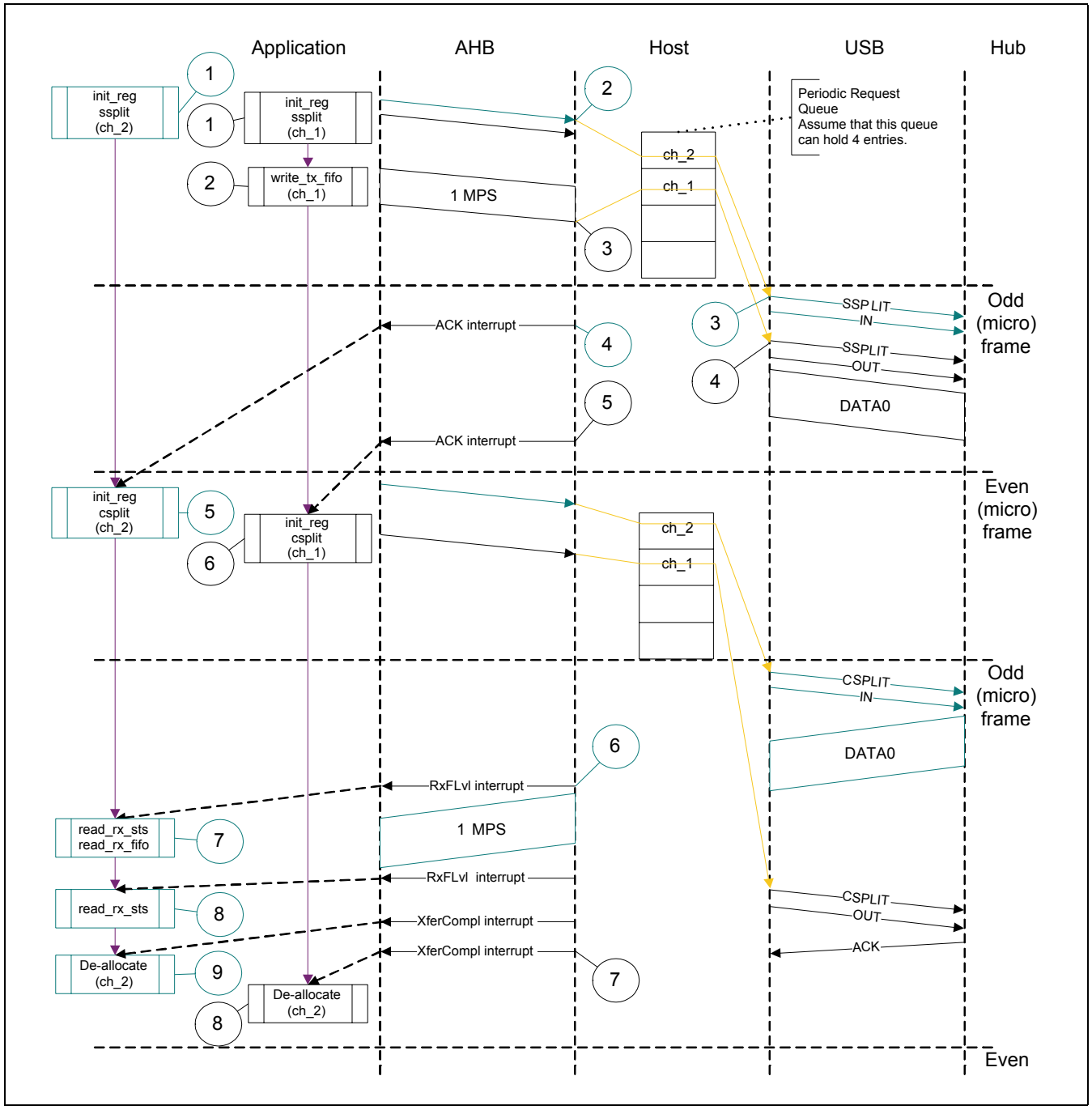
- The application is attempting to transmit one maximum-packet-size packet in an odd microframe.

Normal Interrupt OUT Split Operation

The sequence of operations in *Figure 38-77* on page 1431 (channel 1) is as follows:

1. Initialize and enable channel 1 as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR1.OddFrm bit.
2. Write the packet for channel 1.
3. Along with the last DWORD write of the packet, the USB Host writes an entry to the Periodic Request Queue.
4. The USB Host attempts to send an OUT token in the next odd microframe.
5. The USB Host generates an ACK interrupt as soon as the packet is transmitted successfully.
6. In response to the ACK interrupt, set the HCSPLT1.ComplSplt to send the complete split.
7. The USB Host generates the XferCompl interrupt after successfully completing the complete split transaction.
8. In response to the XferCompl interrupt, de-allocate the channel.

Figure 38-77. Normal Interrupt OUT/IN Split Transactions in Slave Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for interrupt OUT and IN split transactions in Slave mode is shown in *Table 38-78* on page 1432.

Figure 38-78. ISR for Interrupt OUT/IN Split Transactions in Slave Mode

	Interrupt OUT (Split)	Interrupt IN (Split)
Start Split	Unmask (ACK/FrmOvrun) *if (**ACK**)* { Do Complete Split } *else if (**FrmOvrun**)* { Rewind Buffer Pointers Unmask ChHltd Disable Channel (Retry Start Split (in next b_interval - 1 uF)) } *else if (**ChHltd**)* { Mask **ChHltd** De-allocate Channel }	Unmask (ACK/FrmOvrun/DataTglErr) if (ACK) { Do Complete Split } else if (FrmOvrun/DataTglErr) { Rewind Buffer Pointers UnmaskChHltd Disable Channel (Retry Start Split (in next b_interval - 1 uF)) } else if (ChHltd) { Mask ChHltd De-allocate Channel }
Complete Split	Unmask (NAK/XactErr/NYET/STALL/XferCompl/FrmOvrun) *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { Unmask **ChHltd** Disable Channel (Retry Start Split (in next b_interval - 1 uF)) } *else if (**NYET**)* { Retry Complete Split } *else if (**STALL** or **FrmOvrun**)* { Unmask **ChHltd** Disable Channel } *else if (**XactErr**)* { Rewind Buffer Pointers Unmask **ChHltd** Disable Channel (If (HCCHARn.EC == 3), Retry Start Split (in next b_interval - 1 uF)) } *else if (**ChHltd**)* { Mask **ChHltd** De-allocate Channel }	Unmask (NAK/XactErr/NYET/STALL/XferCompl/FrmOvrun/BblErr) *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { UnmaskChHltd Disable Channel (Retry Start Split (in next b_interval - 1 uF)) } *else if (**NYET**)* { Retry Complete Split } *else if (STALL or FrmOvrun or BblErr)* { Unmask **ChHltd** Disable Channel } *else if (**XactErr**)* { Rewind Buffer Pointers Unmask **ChHltd** Disable Channel *(If (HCCHARn.EC == 3), Retry Start Split (in next b_interval - 1 uF))* } *else if (**ChHltd**)* { Mask **ChHltd** De-allocate Channel

Note: The USB Host tracks the error count in EC field for periodic split transactions. If the EC field matches the original programmed error count after XactErr interrupt, the application must treat the XactErr as "ERR response received." The EC field indicates the number of immediate retries that the USB Host has performed before generating the XactErr interrupt.

38.9.5.22 Interrupt IN Split Transactions in Slave Mode

A typical interrupt IN split operation in Slave mode is shown in *Figure 38-77* on page 1431. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one maximum-packet-size packet in an odd microframe.
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet.

Normal Interrupt IN Split Operation

The sequence of operations in *Figure 38-77* on page 1431 (channel 2) is as follows:

1. Initialize and enable channel 2 as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR2.OddFrm bit.
2. The USB Host writes the start split request to the Periodic Request Queue as soon as the HCCHAR2 register is written.
3. The USB Host attempts to send a start split IN token in the next odd microframe.
4. As soon as the IN packet is transmitted, the USB Host generates an ACK interrupt.
5. In response to the ACK interrupt, set the HCSPLT2.CompSplt bit in the next microframe to send the complete split token in the next odd microframe.
6. As soon as the received packet is written to the receive FIFO, the USB Host generates the RxFLvl interrupt.
7. In response to the RxFLvl interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RxFLvl interrupt before reading the receive FIFO and unmask after reading the entire packet.
8. The function generates the RxFLvl interrupt for the transfer completion status entry in the receive FIFO. The application must read the receive packet status and, if the receive packet status is not an IN data packet (GRXSTSR.PktSts != 4'b0010), ignore it.
9. The function generates the XferCompl interrupt as soon as the receive packet status is read. In response to the XferCompl interrupt, read the HCTSIZ2.PktCnt field. If HCTSIZ2.PktCnt != 0, disable the channel (as explained in *Halting a Channel* on page 1398) before re-initializing the channel for the next transfer, if any. If HCTSIZ2.PktCnt == 0, reinstalled the channel for the next transfer. This time, the application must reset the HCCHAR2.OddFrm bit.

Handling Non-ACK Responses

The channel-specific interrupt service routine for an interrupt IN split transaction in Slave mode is shown in *Table 38-78* on page 1432.

38.9.5.23 Interrupt OUT Split Transactions in DMA Mode

A typical interrupt OUT split operation in DMA mode is shown in *Figure 38-79* on page 1434 (see channel 1 [ch_1]). It is assumed that the application is attempting to transmit one packet (1 maximum packet size) in an odd microframe.

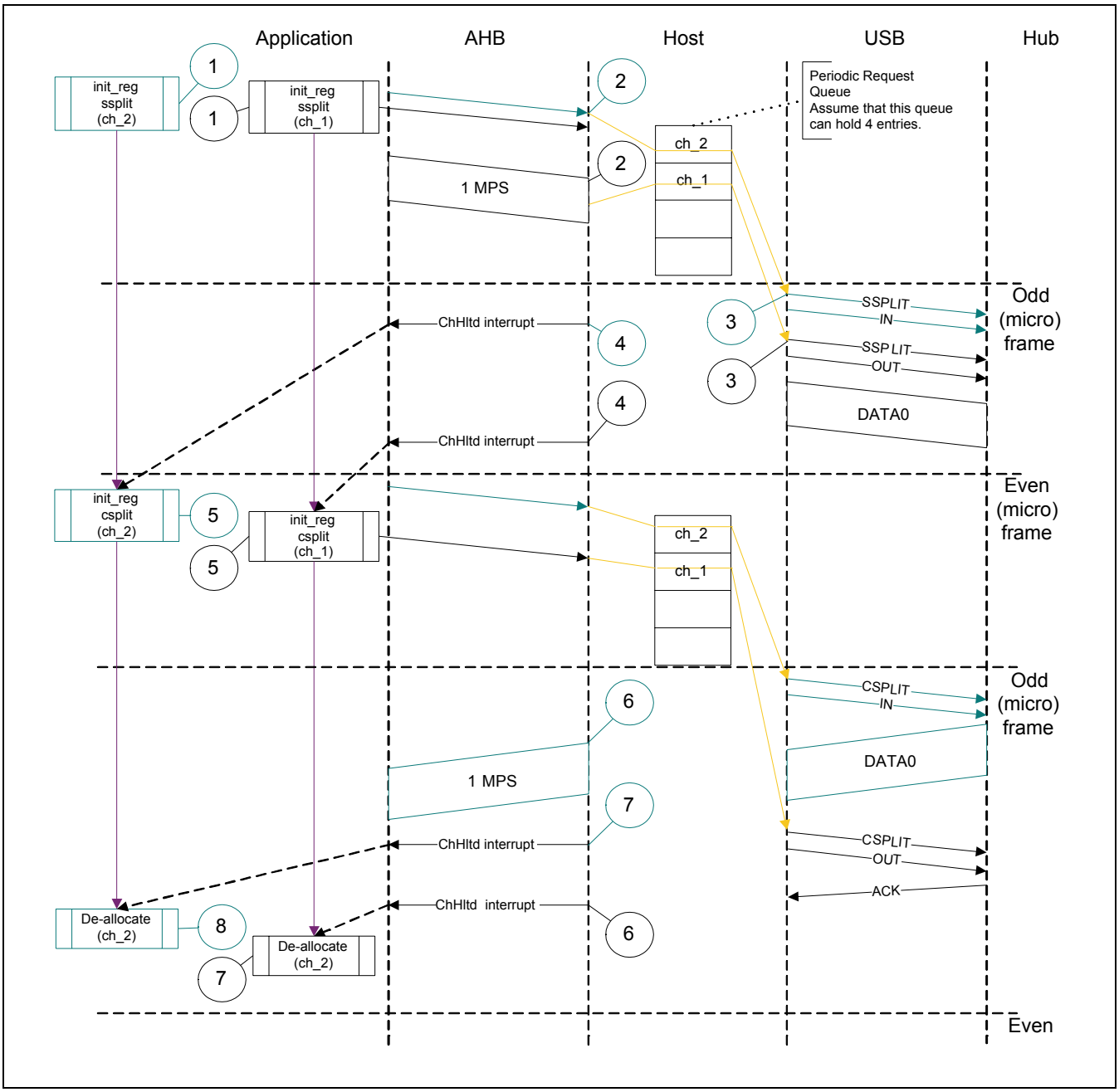
Normal Interrupt OUT Split Operation

The sequence of operations in *Figure 38-79* on page 1434 (channel 1) is as follows:

1. Initialize and enable channel 1 for start split as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR1.OddFrm bit.

2. The USB Host starts reading the packet.
3. The USB Host attempts to send the start split transaction.
4. After successfully transmitting the start split, the USB Host generates the ChHltd interrupt.
5. In response to the ChHltd interrupt, set the HCSPLT1.ComplSplt bit to send the complete split.
6. After successfully completing the complete split transaction, the USB Host generates the ChHltd interrupt.
7. In response to ChHltd interrupt, deallocate the channel.

Figure 38-79. Normal Interrupt OUT/IN Split Transactions in DMA Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for interrupt OUT split transaction in DMA mode is shown in the following table.

Figure 38-80. ISR for Interrupt OUT/IN Split Transactions in DMA Mode

	Interrupt OUT (Split)	Interrupt IN (Split)
Start Split	Unmask (ChHltd) *if (**ChHltd**)* { *if (**ACK**)* { Do Complete Split } *else if (**FrmOvrun**)* { Rewind Buffer Pointers Retry Start Split (in next b_interval - 1 uF) } }	Unmask (ChHltd) if (ChHltd) { if (ACK) { Do Complete Split } else if (FrmOvrun) { Rewind Buffer Pointers Retry Start Split (in next b_interval - 1 uF) } }
Complete Split	Unmask (**ChHltd**) *if (**ChHltd**)* { *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { Retry Start Split (in next b_interval - 1 uF) } *else if (**NYET**)* { Retry Complete Split } *else if (**STALL** or **FrmOvrun**)* { De-allocate Channel } *else if (**XactErr**)* { Rewind Buffer Pointers *if (HCCHARn.EC == 3) // ERR response received* { Retry Start Split (in next b_interval -1 uF) } *else* { De-allocate Channel } } }	Unmask (ChHltd) if (ChHltd) { if (XferCompl) { De-allocate Channel } else if (NAK) { Retry Start Split (in next b_interval -1 uF) } else if (NYET) { Retry Complete Split } else if (STALL or FrmOvrun or BblErr) { De-allocate Channel } else if (XactErr) { Rewind Buffer Pointers if (HCCHARn.EC == 3) // ERR response received { Retry Start Split (in next b_interval - 1 uF) } else { De-allocate Channel } } }

Note: The USB host tracks the error count in the EC field for periodic split transactions. If the EC field matches the original programmed error count after XactErr interrupt, the application must treat the XactErr as an ERR Response Received. The EC field indicates the number of immediate retries the USB Host has performed before generating the XactErr interrupt.

38.9.5.24 Interrupt IN Split Transactions in DMA Mode

A typical interrupt IN split operation in DMA mode is shown in *Figure 38-79* on page 1434. See channel 2 (ch_2).

The assumptions are:

- The application is attempting to receive one maximum-packet-size packet in an odd microframe.
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet.

Normal Interrupt IN Split Operation

The sequence of operations in *Figure 38-79* on page 1434 (channel 2 {ch_2]) is as follows:

1. Initialize and enable channel 2 for start split as explained in *Channel Initialization* on page 1397.
2. The USB Host writes an IN request to the Request queue as soon as channel 2 receives the grant from the arbiter.
3. The USB Host attempts to send the start split IN token in the beginning of the next odd microframe.
4. The USB Host generates the ChHltd interrupt after successfully transmitting the start split IN token.
5. In response to the ChHltd interrupt, set the HCSPLT2.ComplSplt bit to send the complete split.
6. As soon the packet is received successfully, the USB Host starts writing the data to the system memory.
7. The USB Host generates the ChHltd interrupt after transferring the received data to the system memory.
8. In response to the ChHltd interrupt, deallocate or reinstalled the channel for the next start split.

Handling Non-ACK Responses

The channel-specific interrupt service routine for interrupt IN split transaction in DMA mode is shown in *Figure 38-80* on page 1435.

38.9.5.25 Isochronous OUT Split Transactions in Slave Mode

A typical isochronous OUT split operation in Slave mode is shown in *Figure 38-81* on page 1437. See channel 1 (ch_1). The assumptions are:

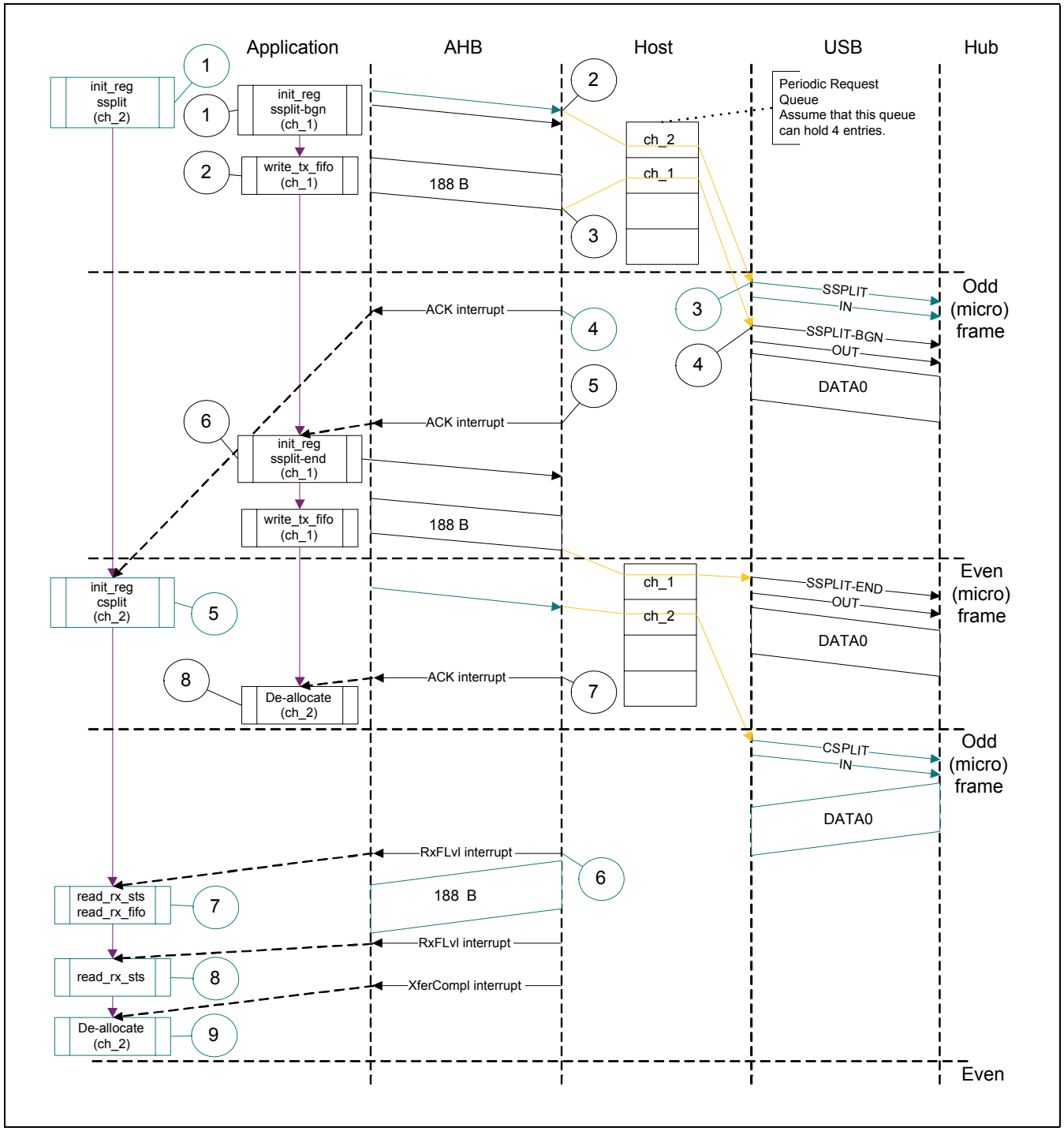
- The application is attempting to transmit a 376-B packet in an odd microframe.

Normal Isochronous OUT Split Operation

The sequence of operations in *Figure 38-81* on page 1437 (channel 1 (ch_1)) is as follows:

1. Initialize and enable channel 1 for start split (begin) as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR1.OddFrm bit. Program the MPS field with 188B.
2. Write the packet for channel 1.
3. Along with the last DWORD write of the packet, the USB Host writes an entry to the Periodic Request Queue.
4. The USB Host attempts to send an OUT token in the next odd microframe.
5. The USB Host generates an ACK interrupt as soon as the packet is transmitted successfully.
6. In response to the ACK interrupt, reinstalled the registers to send the start split (end).
7. The USB Host generates an ACK interrupt after successfully completing the start split (end) transaction.
8. In response to the ACK interrupt, deallocate the channel.

Figure 38-81. Normal Isochronous OUT/IN Split Transactions in Slave Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for isochronous OUT split transaction in Slave mode is shown in the following table.

Figure 38-82. ISR for Isochronous OUT/IN Split Transactions in Slave Mode

	Isochronous OUT (Split	**Isochronous IN (Split)**
Start Split	Unmask (XferCompl) *if (**XferCompl**)* { Do Next Start Split (in next b_interval - 1 uF) } *else if (**FrmOvrun**)* { Do Next Transaction in next frame. }	Unmask (ACK) *if (**ACK**)* { Do Complete Split } *else if (**FrmOvrun**)* { Do Next Transaction in next frame. }
Complete Split	Not Applicable	Unmask (XactErr/NYET/STALL/XferCompl/FrmOvrun/BblErr) *if (**XferCompl**)* { De-allocate Channel } *else if (**NYET**)* { Do Next Complete Split } *else if (*STALL or FrmOvrun or BblErr*)* { Unmask **ChHltd** Disable Channel } *else if (**XactErr**)* { Rewind Buffer Pointers *if (HCCHARn.EC == 3) // ERR response received* { Record ERR error Do Next Start Split (in next frame) } else { Unmask **ChHltd** Disable Channel } *else if (**ChHltd**)* { Mask **ChHltd** De-allocate Channel }

Note: The USB Host keeps track of error count in EC field for periodic split transactions. If the EC field matches the original programmed error count after XactErr interrupt, the application must treat the XactErr as "ERR response received". The EC field indicates the number of immediate retries that the USB Host has performed before generating the XactErr interrupt.

38.9.5.26 Isochronous IN Split Transactions in Slave Mode

A typical isochronous IN split operation in Slave mode is shown in *Figure 38-81* on page 1437. See channel 2 (ch_2). The assumptions are:

- The application is attempting to receive one maximum-packet-size packet in an odd microframe.
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet.

Normal Isochronous IN Split Operation

The sequence of operations in *Figure 38-81* on page 1437 (channel 2) is as follows:

1. Initialize and enable channel 2 for start split as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR2.OddFrm bit.
2. The USB Host writes the start split request to the Periodic Request Queue as soon as the HCCHAR2 register is written.
3. The USB Host attempts to send the start split IN token in the next odd microframe.
4. As soon as the IN packet is transmitted, the USB Host generates an ACK interrupt.
5. In response to the ACK interrupt, set the Do Complete Split bit (HCSPLT2.CompSplt) in the next microframe to send the complete split token in the next odd microframe.
6. As soon as the received packet is written to the receive FIFO, the USB Host generates the RxFLvl interrupt.
7. In response to the RxFLvl interrupt, read the received packet status to determine the number of bytes received, then read the receive FIFO accordingly. The application must mask the RxFLvl interrupt before reading the receive FIFO and unmask it after reading the entire packet.
8. The function generates the RxFLvl interrupt for the transfer completion status entry in the receive FIFO. The application must read the receive packet status and if the receive packet status is not an IN data packet (GRXSTSR.PktSts != 4'b0010), ignore it.
9. The function generates the XferCompl interrupt as soon as the receive packet status is read. In response to the XferCompl interrupt, read the HCTSIZ2.PktCnt field. If HCTSIZ2.PktCnt != 0, disable the channel (as explained in *Halting a Channel* on page 1398) before re-initializing the channel for the next transfer, if any. If HCTSIZ2.PktCnt == 0, reinstalled the channel for the next transfer. This time, the application must reset the HCCHAR2.OddFrm bit.

Handling Non-ACK Responses

The channel-specific interrupt service routine for isochronous IN split transaction in Slave mode is shown in *Table 38-82* on page 1438.

38.9.5.27 Isochronous OUT Split Transactions in DMA Mode

A typical isochronous OUT split operation in DMA mode is shown in *Figure 38-83* on page 1440. See channel 1 (ch_1). The assumption is that the application is attempting to transmit a 376-B packet in an odd microframe.

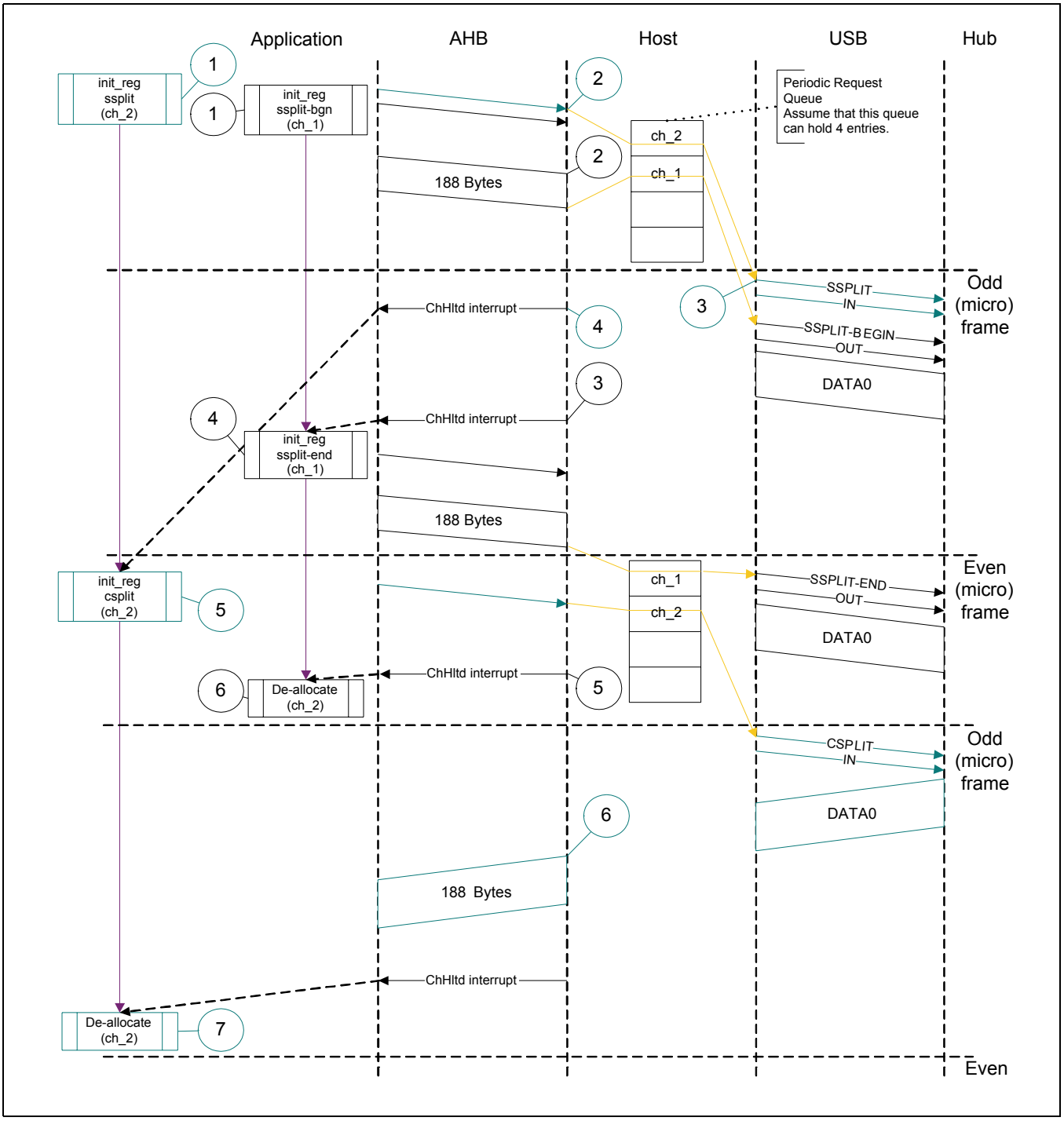
Normal Isochronous OUT Split Operation

The sequence of operations in *Figure 38-83* on page 1440 (channel 1) is as follows:

1. Initialize and enable channel 1 for start split (begin) as explained in *Channel Initialization* on page 1397. The application must set the HCCHAR1.OddFrm bit. Program the MPS field with 188B.
2. The USB Host starts reading the packet.

3. After successfully transmitting the start split (begin), the USB Host generates the ChHltd interrupt.
4. In response to the ChHltd interrupt, reinstalled the registers to send the start split (end).
5. After successfully transmitting the start split (end), the USB Host generates a ChHltd interrupt.
6. In response to the ChHltd interrupt, de-allocate the channel.

Figure 38-83. Normal Isochronous OUT/IN Split Transactions in DMA Mode



Handling Non-ACK Responses

The channel-specific interrupt service routine for an isochronous OUT split transaction in DMA mode is shown in *Figure 38-85* on page 1447.

Figure 38-84. ISR for Isochronous OUT/IN Split Transactions in DMA Mode

	Isochronous OUT (Split	Isochronous IN (Split)
Start Split	Unmask (ChHltd) *if (**ChHltd**)* { *if (**ACK**)* { Do Next Start Split (in next b_interval – 1 uF) } *else if (**FrmOvrun**)* { Do Next Transaction in next frame. } }	Unmask (ChHltd) *if (**ChHltd**)* { *if (**ACK**)* { Do Complete Split } *else if (**FrmOvrun**)* { Rewind Buffer Pointers Retry Start Split (in next b_interval – 1 uF) } }
Complete Split	Not Applicable	Unmask (ChHltd) *if (**ChHltd**)* { *if (**XferCompl**)* { De-allocate Channel } *else if (**NAK**)* { Retry Start Split (in next b_interval – 1 uF) } *else if (**NYET**)* { Do Next Complete Split } *else if (**STALL** or **FrmOvrun** or **BblErr**)* { De-allocate Channel } *else if (**XactErr**)* { Rewind Buffer Pointers *if (HCCHARn.EC = = 3) // ERR response received* { Record ERR error Do Next Start Split (in next frame) } *else* { De-allocate Channel } } }

Note: The USB Host keeps track of error count in EC field for periodic split transactions. If the EC field matches the original programmed error count after XactErr interrupt, the application must treat the XactErr as "ERR response received". The EC field indicates the number of immediate retries that the USB Host has performed before generating the XactErr interrupt.

38.9.5.28 Isochronous IN Split Transactions in DMA Mode

A typical isochronous IN split operation in DMA mode is shown in *Figure 38-83* on page 1440. See channel 2 (ch_2).

The assumptions are:

- The application is attempting to receive one maximum-packet-size packet in an odd microframe.
- The receive FIFO can hold at least one maximum-packet-size packet and two status DWORDs per packet.

Normal Isochronous IN Split Operation

The sequence of operations in *Figure 38-83* on page 1440 (channel 2) is as follows:

1. Initialize and enable channel 2 for start split as explained in *Channel Initialization* on page 1397.
2. The USB Host writes an IN request to the Request queue as soon as channel 2 receives the grant from the arbiter.
3. The USB Host attempts to send the start split IN token in the beginning of the next odd microframe.
4. The USB Host generates the ChHltd interrupt after successfully transmitting the start split IN token.
5. In response to the ChHltd interrupt, set the HCSPLT2.ComplSplt bit to send the complete split.
6. As soon the packet is received successfully, the USB Host starts writing the data to the system memory.
7. The USB Host generates the ChHltd interrupt after transferring the received data to the system memory. In response to the ChHltd interrupt, de-allocate the channel or reinstalled the channel for the next start split.

Handling Non-ACK Responses

The channel-specific interrupt service routine for isochronous IN split transaction in DMA mode is shown in *Figure 38-84* on page 1442.

38.9.6 Selecting the Queue Depth

Choose the Periodic and Non-periodic Request Queue depths carefully to match the number of periodic/non-periodic endpoints accessed.

The Non-periodic Request Queue depth affects the performance of non-periodic transfers. The deeper the queue (along with sufficient FIFO size), the more often the function is able to pipeline non-periodic transfers. If the queue size is small, the function is able to put in new requests only when the queue space is freed up.

The function's Periodic Request Queue depth is critical to performing periodic transfers as scheduled. Select the periodic queue depth, based on the number of periodic transfers scheduled in a microframe. In Slave mode, however, the application must also take into account the disable entry that must be put into the queue. So, if there are two non-high-bandwidth periodic endpoints, the Periodic Request Queue depth must be at least 4. If at least one high-bandwidth Endpoint supported, the queue depth must be 8. If the Periodic Request Queue depth is smaller than the periodic transfers scheduled in a microframe, a frame overrun condition results.

38.9.7 Handling Babble Conditions

USB handles two cases of babble: packet babble and port babble. Packet babble occurs if the Device sends more data than the maximum packet size for the channel. Port babble occurs if the function continues to receive data from the Device at EOF2 (the end of frame 2, which is very close to SOF).

When USB detects a packet babble, it stops writing data into the Rx buffer and waits for the end of packet (EOP). When it detects an EOP, it flushes already-written data in the Rx buffer and generates a Babble interrupt to the application.

When USB detects a port babble, it flushes the RxFIFO and disables the port. The function then generates a Port Disabled Interrupt (GINTSTS.PrtInt, HPRT.PrtEnChng). On receiving this interrupt, the application must determine that this is not due to an over current condition (another cause of the Port Disabled interrupt) by checking HPRT.PrtOvrCurrAct, then perform a soft reset. The function does not send any more tokens after it has detected a port babble condition.

38.10 Device Programming Model

The following sections describe the Device mode programming model.

38.10.1 Endpoint Initialization

The following sections describe initialize an Endpoint.

38.10.1.1 Initialization on USB Reset

1. Set the NAK bit for all OUT endpoints DOEPCTLn.SNAK = 1 (for all OUT endpoints)
2. Unmask the following interrupt bits
   - DAINTMSK.INEP0 = 1 (control 0 IN Endpoint)
   - DAINTMSK.OUTEP0 = 1 (control 0 OUT Endpoint)
   - DOEPMSK.SETUP = 1
   - DOEPMSK.XferCompl = 1
   - DIEPMSK.XferCompl = 1
   - DIEPMSK.TimeOut = 1
3. To transmit or receive data, the Device must initialize more registers as specified in *Device Slave Mode Initialization* on page 1446.
4. Set up the Data FIFO RAM for each of the FIFOs (only if Dynamic FIFO Sizing is enabled)
   - Program the GRXFSIZ Register, to be able to receive control OUT data and setup data. If thresholding is not enabled, at a minimum, this must be equal to one maximum packet size of control Endpoint 0 + 2 DWORDs (for the status of the control OUT data packet) + 10 DWORDs (for setup packets). If thresholding is enabled, at a minimum, this must be equal to 2* (Rx_threshold_length/4 + 1) + 2 DWORDs (for the status of the control OUT data packet) + 10 DWORDs (for setup packets).
   - Program the GNPTXFSIZ Register in Shared FIFO operation or dedicated FIFO size register (depending on the FIFO number chosen) in Dedicated FIFO operation, to be able to transmit control IN data. At a minimum, this must be equal to one maximum packet size of control Endpoint 0. If thresholding is enabled, this can be programmed to less than one maximum packet size.
5. Program the following fields in the Endpoint-specific registers for control OUT Endpoint 0 to receive a SETUP packet
   - DOEPTSIZ0.SetUP Count = 3 (to receive up to 3 back-to-back SETUP packets)
   - In DMA mode, DOEPDMA0 register with a memory address to store any SETUP packets received

At this point, all initialization required to receive SETUP packets is done, except for enabling control OUT Endpoint 0 in DMA mode.

38.10.1.2 Initialization on Enumeration Completion

1. On the Enumeration Done interrupt (GINTSTS.EnumDone, read the DSTS register to determine the enumeration speed.
2. Program the DIEPCTL0.MPS field to set the maximum packet size. This step configures control Endpoint 0. The maximum packet size for a control Endpoint depends on the enumeration speed.
3. In DMA mode, program the DOEPCTL0 register to enable control OUT Endpoint 0, in order to receive a SETUP packet
   - DOEPCTL0. EPEna = 1

At this point, the Device is ready to receive SOF packets and is configured to perform control transfers on control Endpoint 0.

38.10.1.3 Initialization on SetAddress Command

This section describes what the application must do when it receives a SetAddress command in a SETUP packet.

1. Program the DCFG register with the Device address received in the SetAddress command
2. Program the function to send out a status IN packet.

38.10.1.4 Initialization on SetConfiguration/SetInterface Command

This section describes what the application must do when it receives a SetConfiguration or SetInterface command in a SETUP packet.

1. When a SetConfiguration command is received, the application must program the Endpoint registers to configure them with the characteristics of the valid endpoints in the new configuration.
2. When a SetInterface command is received, the application must program the Endpoint registers of the endpoints affected by this command.
3. Some endpoints that were active in the prior configuration or alternate setting are not valid in the new configuration or alternate setting. These invalid endpoints must be deactivated.
4. For details on a particular Endpoint's activation or deactivation, see *Endpoint Activation* on page 1445 and *Endpoint Deactivation* on page 1446.
5. Unmask the interrupt for each active Endpoint and mask the interrupts for all inactive endpoints in the DAINTMSK register.
6. Set up the Data FIFO RAM for each FIFO (only if Dynamic FIFO Sizing is enabled). See *Data FIFO RAM Allocation* on page 1500 for more detail.
7. After all required endpoints are configured, the application must program the function to send a status IN packet.

At this point, the Device function is configured to receive and transmit any type of data packet.

38.10.1.5 Endpoint Activation

This section describes the steps required to activate a Device Endpoint or to configure an existing Device Endpoint to a new type.

1. Program the characteristics of the required Endpoint into the following fields of the DIEPCTLn register (for IN or bidirectional endpoints) or the DOEPCTLn register (for OUT or bidirectional endpoints).

- Maximum Packet Size

- USB Active Endpoint = 1
- Endpoint Start Data Toggle (for interrupt and bulk endpoints)
- Endpoint Type
- TxFIFO Number

  In shared FIFO operation of operation, an interrupt IN Endpoint could be configured as a non periodic Endpoint in applications such as mass storage. The function treats an IN Endpoint as a non-periodic Endpoint, if the DIEPCTLn.TxFNum field is set to 0. Otherwise, a separate periodic FIFO must be allocated using functionConsultant. The number of this FIFO must be programmed into the DIEPCTLn.TxFNum field. Configuring an interrupt IN Endpoint as a non-periodic Endpoint saves the extra periodic FIFO area.

2. Once the Endpoint is activated, the function starts decoding the tokens addressed to that Endpoint and sends out a valid handshake for each valid token received for the Endpoint.

38.10.1.6 Endpoint Deactivation

This section describes the steps required to deactivate an existing Endpoint.

1. In the Endpoint to be deactivated, clear the USB Active Endpoint bit in the DIEPCTLn register (for IN or bidirectional endpoints) or the DOEPCTLn register (for OUT or bidirectional endpoints).
2. Once the Endpoint is deactivated, the function ignores tokens addressed to that Endpoint, resulting in a timeout on the USB.

38.10.1.7 Device Slave Mode Initialization

The application must meet the following conditions to set up the Device function to handle traffic.

- In Slave mode, GINTMSK.NPTxFEmpMsk, and GINTMSK.RxFLvlMsk must be reset.
- In DMA mode, the aforementioned interrupts must be masked.

38.10.2 Operational Model

The following sections describe operational mode programming model.

38.10.2.1 SETUP and OUT Data Transfers

This section describes the internal data flow and application-level operations during data OUT transfers and SETUP transactions.

Packet Read in Slave Mode

This section describes how to read packets (OUT data and SETUP packets) from the receive FIFO in Slave mode.

1. On catching a GINTSTS.RxFLvl interrupt, the application must read the Receive Status Pop register (GRXSTSP).
2. The application can mask the GINTSTS.RxFLvl interrupt by writing to GINTMSK.RxFLvl = 1'b0, until it has read the packet from the receive FIFO.
3. If the received packet's byte count is not 0, the byte count amount of data is popped from the receive Data FIFO and stored in memory. If the received packet byte count is 0, no data is popped from the Receive Data FIFO.
4. The receive FIFO's packet status readout indicates one of the following.

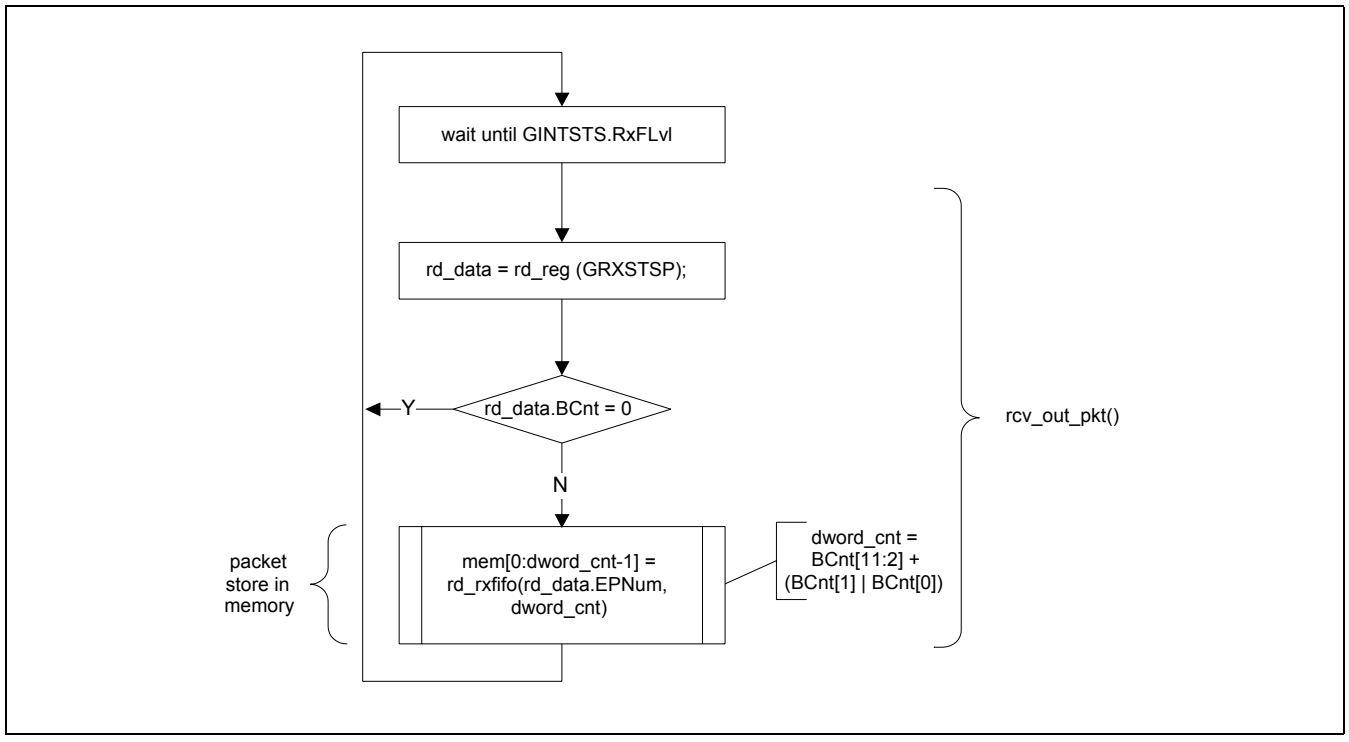
a. Global OUT NAK Pattern: PktSts = Global OUT NAK, BCnt = 11'h000, EPNum = Don't Care (4'h0), DPID = Don't Care (2'b00). This data indicates that the global OUT NAK bit has taken effect.
b. SETUP Packet Pattern: PktSts = SETUP, BCnt = 11'h008, EPNum = Control EP Num, DPID = D0. This data indicates that a SETUP packet for the specified Endpoint is now available for reading from the receive FIFO.
c. Setup Stage Done Pattern: PktSts = Setup Stage Done, BCnt = 11'h0, EPNum = Control EP Num, DPID = Don't Care (2'b00). This data indicates that the Setup stage for the specified Endpoint has completed and the Data stage has started. After this entry is popped from the receive FIFO, the function asserts a Setup interrupt on the specified control OUT Endpoint.
d. Data OUT Packet Pattern: PktSts = DataOUT, BCnt = size of the Received data OUT packet ($0 \leq BCnt \leq 1024$), EPNum = EPNum on which the packet was received, DPID = Actual Data PID.
e. Data Transfer Completed Pattern: PktSts = Data OUT Transfer Done, BCnt = 11'h0, EPNum = OUT EP Num on which the data transfer is complete, DPID = Don't Care (2'b00). This data indicates that a OUT data transfer for the specified OUT Endpoint has completed. After this entry is popped from the receive FIFO, the function asserts a Transfer Completed interrupt on the specified OUT Endpoint.

The encoding for the PktSts is listed in *Rx Status Debug Read/Status Read/Pop Registers (USB0_GRXSTSR/GRXSTSP)* on page 1355.

5. After the data payload is popped from the receive FIFO, the GINTSTS.RxFLvl interrupt must be unmasked.
6. Steps 1–5 are repeated every time the application detects assertion of the interrupt line due to GINTSTS.RxFLvl. Reading an empty receive FIFO can result in undefined function behavior.

The following figure provides a flow chart of the above procedure.

Figure 38-85. Receive FIFO Packet Read in Slave Mode



SETUP Transactions

This section describes how the function handles SETUP packets and the application's sequence for handling SETUP transactions.

Application Requirements

1. To receive a SETUP packet, the DOEPTSIZn.SUPCnt field in a control OUT Endpoint must be programmed to a non-zero value. When the application programs the SUPCnt field to a non-zero value, the function receives SETUP packets and writes them to the receive FIFO, irrespective of the DOEPCTLn.NAK status and DOEPCTLn.EPEna bit setting. The SUPCnt field is decremented every time the control Endpoint receives a SETUP packet. If the SUPCnt field is not programmed to a proper value before receiving a SETUP packet, the function still receives the SETUP packet and decrements the SUPCnt field, but the application possibly is not be able to determine the correct number of SETUP packets received in the Setup stage of a control transfer.
   - DOEPTSIZn.SUPCnt = 3
2. In DMA mode, the OUT Endpoint must also be enabled, to transfer the received SETUP packet data from the internal receive FIFO to the external memory.
   - DOEPCTLn.EPEna = 1'b1
3. The application must always allocate some extra space in the Receive Data FIFO, to be able to receive up to three SETUP packets on a control Endpoint.
   - The space to be Reserved is (4 * *n*) + 6 DWORDs, where *n* is the number of control endpoints supported by the Device. Three DWORDs are required for the first SETUP packet, 1 DWORD is required for the Setup Stage Done DWORD, and 6 DWORDs are required to store two extra SETUP packets among all control endpoints.
   - 3 DWORDs per SETUP packet are required to store 8B of SETUP data and 4B of SETUP status (Setup Packet Pattern). The function reserves this space in the receive data
   - FIFO to write SETUP data only, and never uses this space for data packets.
4. In Slave mode, the application must read the 2 DWORDs of the SETUP packet from the receive FIFO. In DMA mode, the function writes the 2 DWORDs of SETUP data to the memory.
5. The application must read and discard the Setup Stage Done DWORD from the receive FIFO.

Internal Data Flow

1. When a SETUP packet is received, the function writes the received data to the receive FIFO, without checking for available space in the receive FIFO and irrespective of the Endpoint's NAK and Stall bit settings.

- The function internally sets the IN NAK and OUT NAK bits for the control IN/OUT endpoints on which the SETUP packet was received.

2. For every SETUP packet received on the USB, 3 DWORDs of data is written to the receive FIFO, and the SUPCnt field is decremented by 1.
   - The first DWORD contains control information used internally by the function
   - The second DWORD contains the first 4B of the SETUP command
   - The third DWORD contains the last 4B of the SETUP command
3. When the Setup stage changes to a Data IN/OUT stage, the function writes an entry (Setup Stage Done DWORD) to the receive FIFO, indicating the completion of the Setup stage.
4. On the AHB side, SETUP packets are emptied either by the DMA or the application. In DMA mode, the SETUP packets (2 DWORDs) are written to the memory location programmed in the DOEPDMAn register, only if the Endpoint is enabled. If the Endpoint is not enabled, the data remains in the receive FIFO until the enable bit is set.

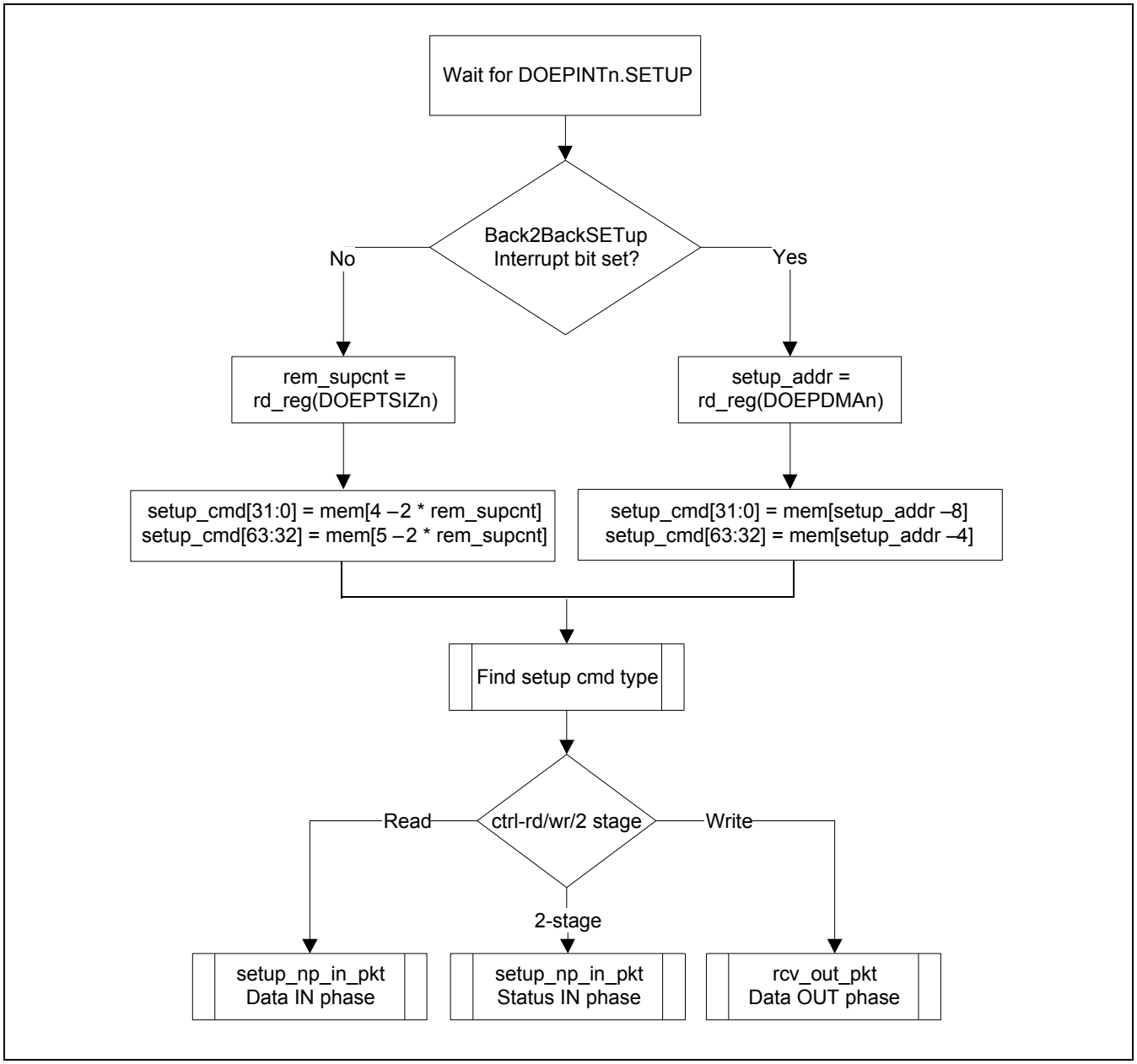
5. When either the DMA or the application pops the Setup Stage Done DWORD from the receive FIFO, the function interrupts the application with a DOEPINTn.SETUP interrupt, indicating it can process the received SETUP packet.
    - The function clears the Endpoint enable bit for control OUT endpoints.

Application Programming Sequence

1. Program the DOEPTSIZn register.
    - DOEPTSIZn.SUPCnt = 3
2. In DMA mode, program the DOEPDMAn register and DOEPCTLn register with the Endpoint characteristics and set the Endpoint Enable bit (DOEPCTLn. EPEna).
    - Endpoint Enable = 1
3. In Slave mode, wait for the GINTSTS.RxFLvl interrupt and empty the data packets from the receive FIFO, as explained in *Packet Read in Slave Mode* on page 1446. This step can be repeated many times.
4. Assertion of the DOEPINTn.SETUP interrupt marks a successful completion of the SETUP Data Transfer.
    - On this interrupt, the application must read the DOEPTSIZn register to determine the number of SETUP packets received and process the last received SETUP packet.
    - In DMA mode, the application must also determine if the interrupt bit DOEPINTn.Back2BackSETup is set. This bit is set if the function has received more than three back-to-back SETUP packets. If this is the case, the application must ignore the DOEPTSIZn.SUPCnt value and use the DOEPDMAn directly to read out the last SETUP packet received. DOEPDMAn-8 provides the pointer to the last valid SETUP data.

The following figure charts this flow.

Figure 38-86. Processing a SETUP Packet



Handling More Than Three Back-to-Back SETUP Packets

Per the USB 2.0 specification, normally, during a SETUP packet error, a Host does not send more than three back-to-back SETUP packets to the same Endpoint. However, the USB 2.0 specification does not limit the number of back-to-back SETUP packets a Host can send to the same Endpoint. When this condition occurs, the USB function generates an interrupt (DOEPINTn.Back2BackSETup). In DMA mode, the function also rewinds the DMA address for that Endpoint (DOEPDMAn) and overwrites the first SETUP packet in system memory with the fourth, second with the fifth, and so on. If the Back2BackSETup interrupt is asserted, the application must read the OUT Endpoint DMA register (DOEPDMAn) to determine the final SETUP data in system memory.

In DMA mode, the application can mask the Back2BackSETup interrupt, but after receiving the DOEPINT.SETUP interrupt, the application can read the DOEPINT.Back2BackSETup interrupt bit. In Slave mode, the application can use the GINTSTS.RxFLvl interrupt to read out the SETUP packets from the FIFO whenever the function receives the SETUP packet.

Setting the Global OUT NAK

Internal Data Flow

1. When the application sets the Global OUT NAK (DCTL.SGOUTNak), the function stops writing data, except SETUP packets, to the receive FIFO. Irrespective of the space availability in the receive FIFO, non-isochronous OUT tokens receive a NAK handshake response, and the function ignores isochronous OUT data packets
2. The function writes the Global OUT NAK pattern to the receive FIFO. The application must reserve enough receive FIFO space to write this data pattern. See*Data FIFO RAM Allocation* on page 1500.
3. When either the function (in DMA mode) or the application (in Slave mode) pops the Global OUT NAK pattern DWORD from the receive FIFO, the function sets the GINTSTS.GOUTNakEff interrupt.
4. Once the application detects this interrupt, it can assume that the function is in Global OUT NAK mode. The application can clear this interrupt by clearing the DCTL.SGOUTNak bit.

Application Programming Sequence

1. To stop receiving any kind of data in the receive FIFO, the application must set the Global OUT NAK bit by programming the following field.
   - DCTL.SGOUTNak = 1
2. Wait for the assertion of the interrupt GINTSTS.GOUTNakEff. When asserted, this interrupt indicates that the function has stopped receiving any type of data except SETUP packets.
3. The application can receive valid OUT packets after it has set DCTL.SGOUTNak and before the function asserts the GINTSTS.GOUTNakEff interrupt.
4. The application can temporarily mask this interrupt by writing to the GINTMSK.GINNakEffMsk bit.
   - GINTMSK.GINNakEffMsk = 0
5. Whenever the application is ready to exit the Global OUT NAK mode, it must clear the DCTL.SGOUTNak bit. This also clears the GINTSTS.GOUTNakEff interrupt.
   - DCTL.CGOUTNak = 1
6. If the application has masked this interrupt earlier, it must be unmasked as follows:
   - GINTMSK.GINNakEffMsk = 1

Disabling an OUT Endpoint

The application must use this sequence to disable an OUT Endpoint that it has enabled.

Application Programming Sequence

1. Before disabling any OUT Endpoint, the application must enable Global OUT NAK mode in the function, as described in *Setting the Global OUT NAK* on page 1451.
   - DCTL.DCTL.SGOUTNak = 1
2. Wait for the GINTSTS.GOUTNakEff interrupt

3. Disable the required OUT Endpoint by programming the following fields
   - DOEPCTLn.EPDisable = 1
   - DOEPCTLn.SNAK = 1
4. Wait for the DOEPINTn.EPDisabled interrupt, which indicates that the OUT Endpoint is completely disabled. When the EPDisabled interrupt is asserted, the function also clears the following bits.
   - DOEPCTLn.EPDisable = 0
   - DOEPCTLn.EPEnable = 0
5. The application must clear the Global OUT NAK bit to start receiving data from other non-disabled OUT endpoints.
   - DCTL.SGOUTNak = 0

Generic Non-Isochronous OUT Data Transfers Without Thresholding

This section describes a regular non-isochronous OUT data transfer (control, bulk, or interrupt).

Application Requirements

1. Before setting up an OUT transfer, the application must allocate a buffer in the memory to accommodate all data to be received as part of the OUT transfer.
2. For OUT transfers, the Transfer Size field in the Endpoint's Transfer Size register must be a multiple of the maximum packet size of the Endpoint, adjusted to the DWORD boundary
   - transfer size[epnum] = n * (mps[epnum] + 4 – (mps[epnum] mod 4))
   - packet count[epnum] = n
   - n > 0
3. In DMA mode, the function stores a received data packet in the memory, always starting on a DWORD boundary. If the maximum packet size of the Endpoint is not a multiple of four, the function inserts byte pads at end of a maximum-packet-size packet up to the end of the DWORD.
4. On any OUT Endpoint interrupt, the application must read the Endpoint's Transfer Size register to calculate the size of the payload in the memory. The received payload size can be less than the programmed transfer size.
   - Payload size in memory = application-programmed initial transfer size – function updated final transfer size
   - Number of USB packets in which this payload was received = application-programmed initial packet count – function updated final packet count

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the Endpoint-specific registers, clear the NAK bit, and enable the Endpoint to receive the data.
2. Once the NAK bit is cleared, the function starts receiving data and writes it to the receive FIFO, as long as there is space in the receive FIFO. For every data packet received on the USB, the data packet and its status are written to the receive FIFO. Every packet (maximum packet size or short packet) written to the receive FIFO decrements the Packet Count field for that Endpoint by 1.
   - OUT data packets received with Bad Data CRC are flushed from the receive FIFO automatically.
   - After sending an ACK for the packet on the USB, the function discards non-isochronous OUT data packets that the Host, which cannot detect the ACK, re-sends. The application does not detect multiple back-to-back data OUT packets on the same Endpoint with the same data PID. In this case the packet count is not decremented.

- If there is no space in the receive FIFO, isochronous or non-isochronous data packets are ignored and not written to the receive FIFO. Additionally, non-isochronous OUT tokens receive a NAK handshake reply.
- In all the above three cases, the packet count is not decremented because no data is written to the receive FIFO.

3. When the packet count becomes 0 or when a short packet is received on the Endpoint, the NAK bit for that Endpoint is set. Once the NAK bit is set, the isochronous or non-isochronous data packets are ignored and not written to the receive FIFO, and non-isochronous OUT tokens receive a NAK handshake reply.
4. After the data is written to the receive FIFO, either the application (in Slave mode) or the function's DMA engine (in External or Internal DMA mode), reads the data from the receive FIFO and writes it to external memory, one packet at a time per Endpoint.
5. At the end of every packet write on the AHB to external memory, the transfer size for the Endpoint is decremented by the size of the written packet.
6. The OUT Data Transfer Completed pattern for an OUT Endpoint is written to the receive FIFO on one of the following conditions.
   - The transfer size is 0 and the packet count is 0.
   - The last OUT data packet written to the receive FIFO is a short packet (0 ≤ packet size < maximum packet size).
7. When either the application or DMA pops this entry (OUT Data Transfer Completed), a Transfer Completed interrupt is generated for the Endpoint and the Endpoint enable is cleared.

Application Programming Sequence

1. Program the DOEPTSIZn register for the transfer size and the corresponding packet count. Additionally, in DMA mode, program the DOEPDMAn register.
2. Program the DOEPCTLn register with the Endpoint characteristics, and set the Endpoint Enable and ClearNAK bits.
   - DOEPCTLn.EPEna = 1
   - DOEPCTLn.CNAK = 1
3. In Slave mode, wait for the GINTSTS.Rx StsQ level interrupt and empty the data packets from the receive FIFO as explained in *Packet Read in Slave Mode* on page 1446.
   - This step can be repeated many times, depending on the transfer size.
4. Asserting the DOEPINTn.XferCompl interrupt marks a successful completion of the non-isochronous OUT data transfer.

Generic Non-Isochronous OUT Data Transfer With Thresholding

This section describes a regular non-ISO OUT data transfer (Control/Bulk/Intr) when thresholding is enabled.

Application Requirements

Application requirements are the same as without thresholding.

Internal Data Flow

1. The application should set the transfer size and packet count fields in the Endpoint specific registers, clear the NAK bit and enable the Endpoint to receive the data.

2. Once the NAK bit is cleared, the function starts receiving the data and writes it into the receive FIFO, as long as there is threshold amount of space in the receive FIFO. For every threshold amount of data received on the USB, the data packet and the threshold status are written into the receive FIFO. At the end of a packet, a last threshold status and also a data update status is written into the receive FIFO. If it was the last packet of the transfer, then a transfer complete status is also written into the receive FIFO. On every packet (mps sized or short packet) written into the receive FIFO, the packet count field for that Endpoint is decremented by one.
   - OUT data packets received with Bad Data CRC are flushed out of the receive FIFO automatically. The function also rewinds the DMA pointers internally.
   - Non-ISO OUT data packet re-sent by the Host, because the ACK was not seen by the Host, are discarded by the function, after sending an ACK for the packet on the USB. The application does not see multiple back to back data OUT packets on the same Endpoint, with the same data PID. In this case the packet count is not decremented.
   - If there is no space for at least threshold amount of data in the receive FIFO, the ISO/non-ISO data packets are ignored and not written into the receive FIFO. In addition, the non-ISO OUT tokens are responded with NAK handshake.
   - If the function sees an overflow case (no space in the fifo in the middle of a packet reception), then the function stops writing the remaining data into the fifo and sends a NAK handshake on the USB. The function rewinds the fifo pointer to the threshold boundary, so that the portion of the threshold data that is in the fifo is flushed out. The function also rewinds the DMA pointers. The function also sets DOEPINTn.OutPktErr (This interrupt bit is mainly used for debug purpose).

   In all the above cases, the packet count is not decremented because no data is written into the receive FIFO.

   In high speed, after the function has received a packet, the function sends a NYET handshake if the function does not find threshold amount of free space available in the FIFO.
3. When the packet count becomes 0 or when a short packet is received on the Endpoint, the NAK bit for that Endpoint is set. Once the NAK bit is set, the ISO/non-ISO data packets are ignored and not written into the receive FIFO and the non-ISO OUT tokens are responded with a NAK handshake.
4. The DMA engine transfers data from the receive FIFO to the system memory as soon as it sees one threshold amount of data in the FIFO.
5. At the end of every packet write on the AHB into the external memory, the transfer size for the Endpoint.
6. The OUT Data Transfer Complete pattern for an OUT Endpoint is written into the receive FIFO, on one of the following conditions:
   - The last threshold is written into FIFO and the packet count is decremented to 0.
   - If it is a short packet and the function sees the end of packet within a threshold.
7. When this entry (OUT Data Transfer Complete) is popped out by the DMA engine, Transfer Complete interrupt for the Endpoint is generated and the Endpoint enable is cleared.
8. The "Rewind OUT Data Transfer" pattern is written into the receive FIFO, on one of the following conditions:
   - On seeing Overflow condition.
   - On seeing a CRC error.
9. When this entry (Rewind OUT Data Transfer) is popped out by the DMA engine, it does the DMA pointer rewind.

Application Programming Sequence

This sequence is the same as in non thresholding case.

Generic Isochronous OUT Data Transfer Without Thresholding

This section describes a regular isochronous OUT data transfer.

Application Requirements

1. All the application requirements for non-isochronous OUT data transfers also apply to isochronous OUT data transfers
2. For isochronous OUT data transfers, the Transfer Size and Packet Count fields must always be set to the number of maximum-packet-size packets that can be received in a single microframe and no more. Isochronous OUT data transfers cannot span more than 1 microframe.
   - $1 \leq$ packet count[epnum] ≤ 3
3. In Slave mode, when isochronous OUT endpoints are supported in the Device, the application must read all isochronous OUT data packets from the receive FIFO (data and status) before the end of the periodic frame (GINTSTS.EOPF interrupt). In DMA mode, the application must guarantee enough bandwidth to allow emptying the isochronous OUT data packet from the receive FIFO before the end of each periodic frame.
4. To receive data in the following frame/microframe, an isochronous OUT Endpoint must be enabled after the GINTSTS.EOPF and before the GINTSTS.SOF.

Internal Data Flow

1. The internal data flow for isochronous OUT endpoints is the same as that for non-isochronous OUT endpoints, but for a few differences.
2. When an isochronous OUT Endpoint is enabled by setting the Endpoint Enable and clearing the NAK bits, the Even/Odd frame/microframe bit must also be set appropriately. The function receives data on a isochronous OUT Endpoint in a particular microframe only if the following condition is met.
   - DOEPCTLn.Even/Odd microframe = DSTS.SOFFN[0]
3. When either the application or the external/internal DMA completely reads an isochronous OUT data packet (data and status) from the receive FIFO, the function updates the DOEPTSIZn.Received DPID field with the data PID of the last isochronous OUT data packet read from the receive FIFO.

Application Programming Sequence

1. Program the DOEPTSIZn register for the transfer size and the corresponding packet count. When in DMA mode, also program the DOEPDMAn register.
2. Program the DOEPCTLn register with the Endpoint characteristics and set the Endpoint Enable, ClearNAK, and Even/Odd frame/microframe bits.
   - Endpoint Enable = 1
   - CNAK = 1
   - Even/Odd frame/microframe = (0: Even/1: Odd)
3. In Slave mode, wait for the GINTSTS.Rx StsQ level interrupt and empty the data packets from the receive FIFO as explained in *Packet Read in Slave Mode* on page 1446.
   - This step can be repeated many times, depending on the transfer size.
4. The assertion of the DOEPINTn.XferCompl interrupt marks the completion of the isochronous OUT data transfer. This interrupt does not necessarily mean that the data in memory is good.
   - This interrupt can not always be detected for isochronous OUT transfers. Instead, the application can detect the GINTSTS.incomplete Isochronous OUT data interrupt. See *Incomplete Isochronous OUT Data Transfers* on page 1457 for more details

5. Read the DOEPTSIZn register to determine the size of the received transfer and to determine the validity of the data received in the microframe. The application must treat the data received in memory as valid only if one of the following conditions is met.
   - DOEPTSIZn.RxDPID = D0 and the number of USB packets in which this payload was received = 1
   - DOEPTSIZn.RxDPID = D1 and the number of USB packets in which this payload was received = 2
   - DOEPTSIZn.RxDPID = D2 and the number of USB packets in which this payload was received = 3
     – The number of USB packets in which this payload was received = App Programmed Initial Packet Count – function Updated Final Packet Count

Generic Isochronous OUT Data Transfer With Thresholding

This section describes a regular isochronous OUT data transfer.

Application Requirements

There is no change in this section from a non-thresholding mode.

Internal Data Flow

1. The internal data flow for isochronous OUT Endpoints when thresholding is enabled, is the same as that for the non isochronous OUT endpoints when thresholding is enabled, but for a few differences.
2. If MAC sees an overflow condition when writing a packet, it stops writing. The current threshold amount of data that is being written into the receive FIFO is flushed out at the end of packet. This eventually results in GINTSTS.incomplete ISO OUT Data interrupt. Refer to the section Incomplete isochronous OUT Data Transfers, for more details.
3. If MAC sees a CRC error for the receiving packet, the last threshold being written into the receive FIFO is flushed out. This will eventually result in GINTSTS.incomplete isochronous OUT Data interrupt. Refer to the section Incomplete isochronous OUT Data Transfers, for more details.
4. Assertion of DOEPINTn.XferCompl interrupt marks a completion of the isochronous OUT Data Transfer. This interrupt may not necessarily mean that data in the memory is good data.
5. Read the DOEPTSIZn register, to find out the size of the received transfer and to find out the validity of the data received in the microframe. The application should treat the data received into the memory as valid only if one of the following conditions is met. Invalid data packets may be discarded by the application.
   - DOEPTSIZn.RxDPID = D0 and Number of USB Packets in which this payload was received = 1
   - DOEPTSIZn.RxDPID = D1 and Number of USB Packets in which this payload was received = 2
   - DOEPTSIZn.RxDPID = D2 and Number of USB Packets in which this payload was received = 3

   Number of USB Packets in which this payload was received = App Programmed Initial Packet Count – function Updated Final Packet Count

   The application can discard invalid data packets.

Incomplete Isochronous OUT Data Transfers

This section describes the application programming sequence when isochronous OUT data packets are dropped inside the function.

Internal Data Flow

1. For isochronous OUT endpoints, the DOEPINTn.XferCompl interrupt possibly is not always asserted. If the function drops isochronous OUT data packets, the application could fail to detect the DOEPINTn.XferCompl interrupt under the following circumstances.
   - When the receive FIFO cannot accommodate the complete ISO OUT data packet, the function drops the received ISO OUT data. In thresholding this is the same as overflow.
   - When the isochronous OUT data packet is received with CRC errors
   - When the isochronous OUT token received by the function is corrupted
   - When the application is very slow in reading the data from the receive FIFO
2. When the function detects an end of periodic frame before transfer completion to all isochronous OUT endpoints, it asserts the GINTSTS.incomplete Isochronous OUT data interrupt, indicating that a DOEPINTn.XferCompl interrupt is not asserted on at least one of the isochronous OUT endpoints. At this point, the Endpoint with the incomplete transfer remains enabled, but no active transfers remains in progress on this Endpoint on the USB.

Application Programming Sequence

1. Asserting the GINTSTS.incomplete Isochronous OUT data interrupt indicates that in the current microframe, at least one isochronous OUT Endpoint has an incomplete transfer.
   - If this occurs because isochronous OUT data is not completely emptied from the Endpoint, the application must ensure that the DMA or application empties all isochronous OUT data (data and status) from the receive FIFO before proceeding.
   - When all data is emptied from the receive FIFO, the application can detect the DOEPINTn.XferCompl interrupt. In this case, the application must re-enable the Endpoint to receive isochronous OUT data in the next microframe, as described in *Generic Non-Isochronous OUT Data Transfer With Thresholding* on page 1453.
2. When it receives a GINTSTS.incomplete Isochronous OUT data interrupt, the application must read the control registers of all isochronous OUT endpoints (DOEPCTLn) to determine which endpoints had an incomplete transfer in the current microframe. An Endpoint transfer is incomplete if both the following conditions are met.
   - DOEPCTLn.Even/Odd microframe bit = DSTS.SOFFN[0]
   - DOEPCTLn.Endpoint Enable = 1
3. The previous step must be performed before the GINTSTS.SOF interrupt is detected, to ensure that the current microframe number is not changed.
4. For isochronous OUT endpoints with incomplete transfers, the application must discard the data in the memory and disable the Endpoint by setting the DOEPCTLn.Endpoint Disable bit.
5. Wait for the DOEPINTn.Endpoint Disabled interrupt and enable the Endpoint to receive new data in the next microframe as explained in *Generic Non-Isochronous OUT Data Transfer With Thresholding* on page 1453.
   - Because the function can take some time to disable the Endpoint, the application possibly is not able to receive the data in the next microframe after receiving bad isochronous data.

Stalling a Non-Isochronous OUT Endpoint

This section describes how the application can stall a non-isochronous Endpoint.

1. Put the function in the Global OUT NAK mode, as described in *Setting the Global OUT NAK* on page 1451.
2. Disable the required Endpoint, as described in *Disabling an OUT Endpoint* on page 1451.
   - When disabling the Endpoint, instead of setting the DOEPCTL.SNAK bit, set DOEPCTL.STALL = 1.

   Note: The Stall bit always takes precedence over the NAK bit.
3. When the application is ready to end the STALL handshake for the Endpoint, the DOEPCTLn.STALL bit must be cleared.
4. If the application is setting or clearing a STALL for an Endpoint due to a SetFeature.Endpoint Halt or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control Endpoint.

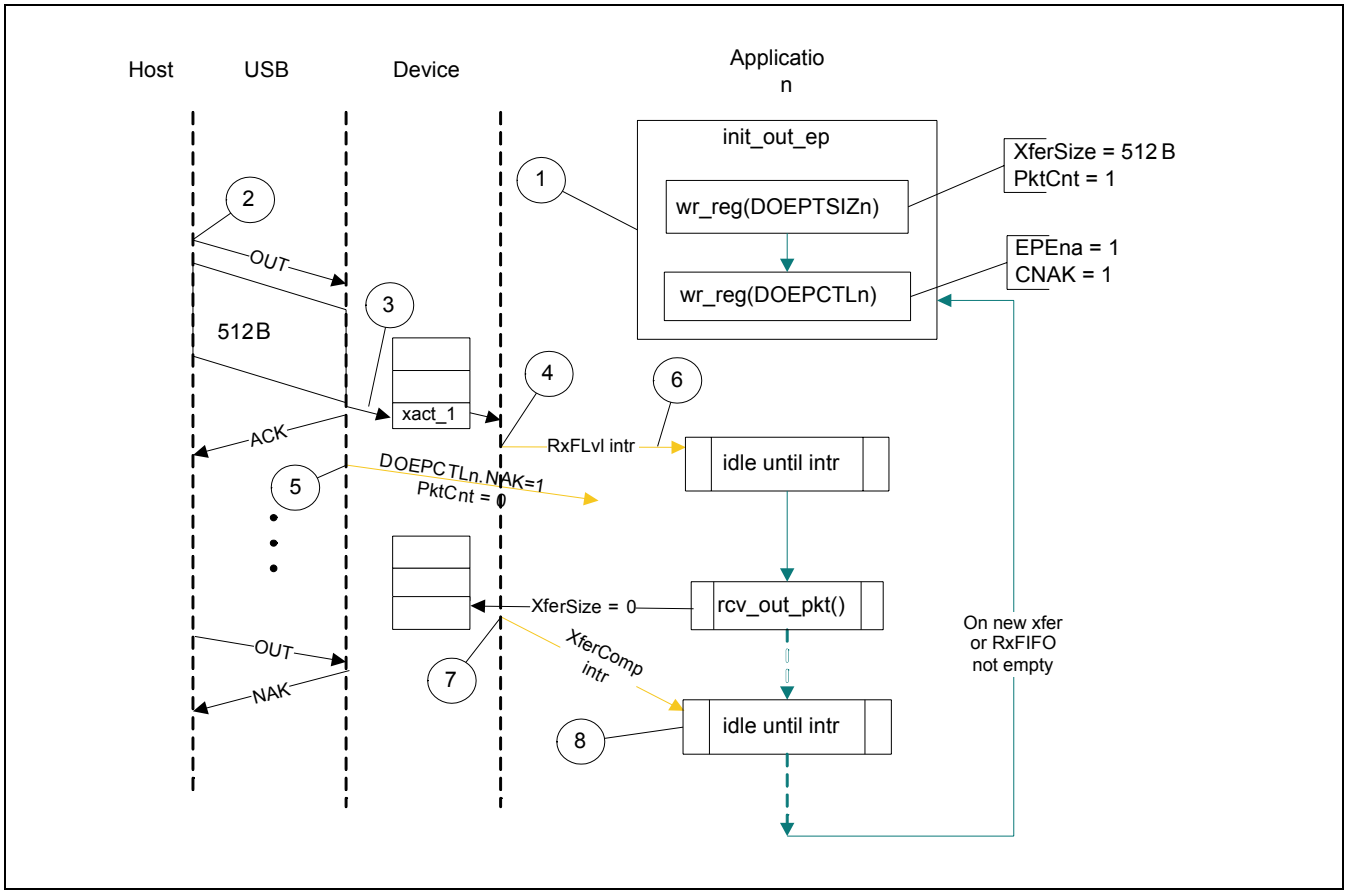
Examples:

This section describes and depicts some fundamental transfer types and scenarios.

Slave Mode Bulk OUT Transaction

The following figure depicts the reception of a single Bulk OUT Data packet from the USB to the AHB and describes the events involved in the process.

Figure 38-87. Slave Mode Bulk OUT Transaction



1. After a SetConfiguration/SetInterface command, the application initializes all OUT endpoints by setting DOEPCTLn.CNAK = 1 and DOEPCTLn.EPEna = 1, and setting a suitable XferSize and PktCnt in the DOEPTSIZn register.
2. Host attempts to send data (OUT token) to an Endpoint.
3. When the function receives the OUT token on the USB, it stores the packet in the RxFIFO because space is available there.
4. After writing the complete packet in the RxFIFO, the function then asserts the GINTSTS.RxFLvl interrupt.
5. On receiving the PktCnt number of USB packets, the function sets the NAK bit for this Endpoint internally to prevent it from receiving any more packets.
6. The application processes the interrupt and reads the data from the RxFIFO.
7. When the application has read all the data (equivalent to XferSize), the function generates a DOEPINTn.XferCompl interrupt.
8. The application processes the interrupt and uses the setting of the DOEPINTn.XferCompl interrupt bit to determine that the intended transfer is complete.

38.10.2.2 IN Data Transfers

This section describes IN data transfer details.

Non-periodic Packet Write in Slave Mode: Shared FIFO

This section describes how an application writes a data packet to the Non-periodic Transmit FIFO in non-periodic transmit FIFO in shared FIFO operation (OTG_EN_DED_TX_FIFO=0).

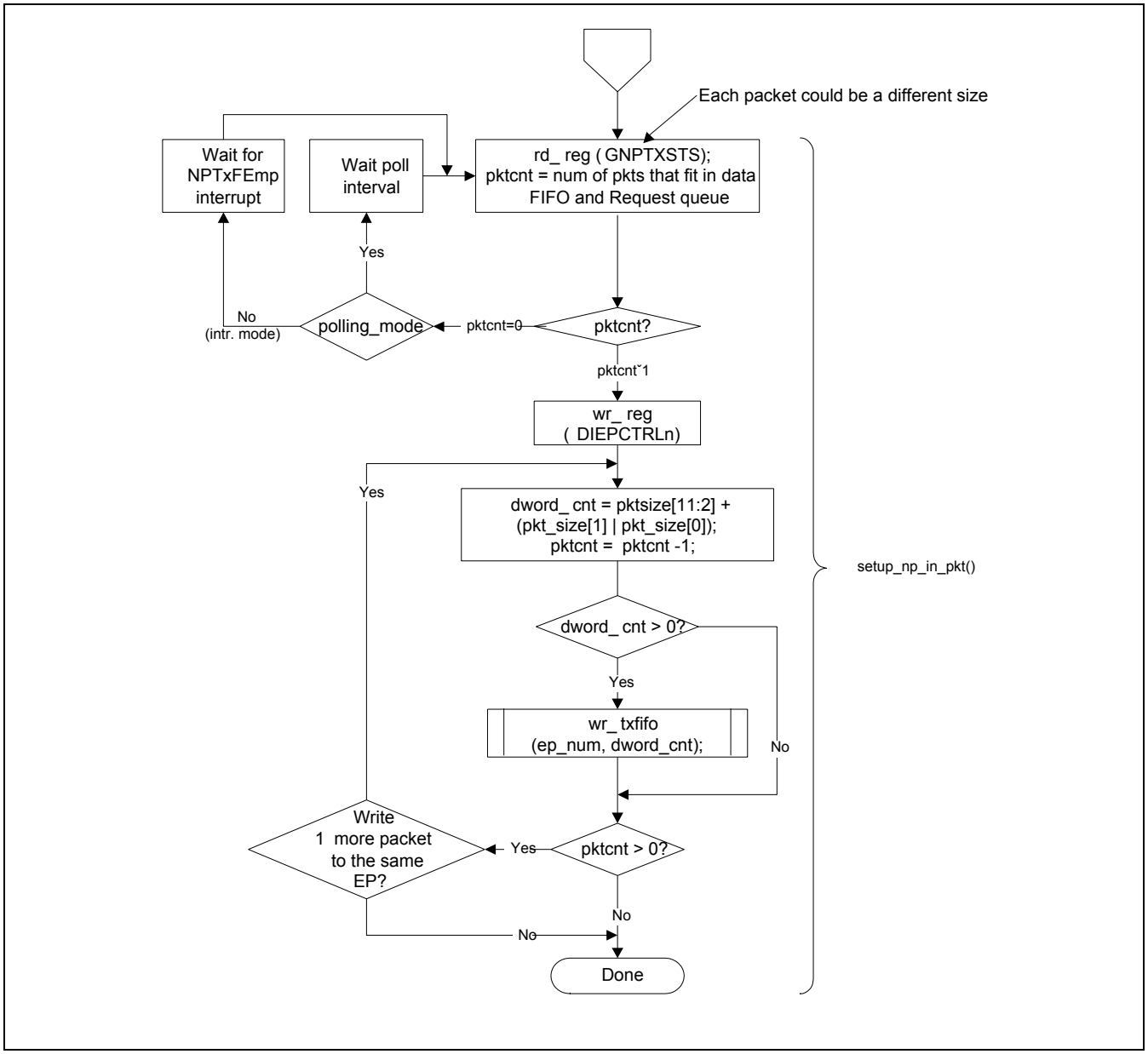
1. The application can either operate in Polling mode or Interrupt mode.

- In Polling mode, the application monitors the Non-periodic Transmit FIFO's status by reading the GNPTXSTS register to determine if enough space is available in the Data FIFO and the Request queue.
- In Interrupt mode, the application waits for the Non-periodic Transmit FIFO Empty interrupt (GINTSTS.NPTxFEmp), then reads the GNPTXSTS register to determine if enough space is available in the Data FIFO and the Request queue.
- To write a single non-zero length data packet, there must be 1 empty space available in the Request queue and enough space to write the entire packet in the Data FIFO.
- To write a single zero length data packet, there must be 1 empty space available in the Request queue, but no space is required in the Data FIFO.

2. When the application determines, using one of the aforementioned methods, that there is enough space to write a non-periodic transmit packet, the application must first write to the Endpoint control register before writing the data to the Data FIFO. The application typically must perform a read-modify-write on the DIEPCTLn to avoid modifying the contents of the register beyond setting the Endpoint Enable bit.
3. If space is available, the application can write multiple packets for the same Endpoint to the Non-periodic Transmit FIFO.

This procedure is depicted in the following figure.

Figure 38-88. Non-periodic Data Packet Write in Slave Mode (Shared FIFO)



Periodic Packet Write in Slave Mode: Shared FIFO

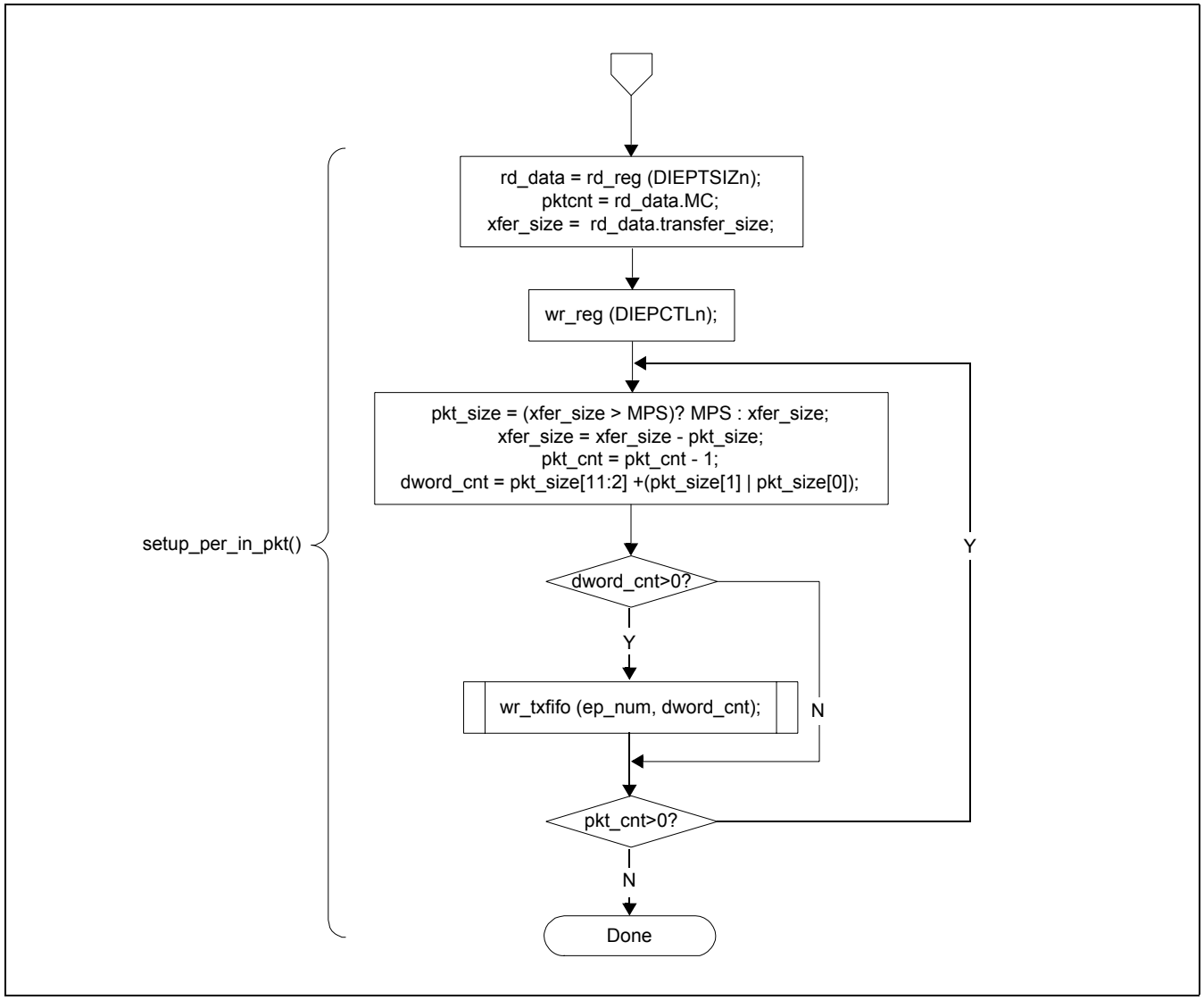
This section describes the procedure for the application to write periodic data to the Periodic Transmit FIFO for 1 microframe.

1. Unlike non-periodic data writes, the application need not check for space availability in the periodic FIFO. When the application writes a packet to the periodic FIFO, it automatically overwrites the FIFO's existing contents.
2. The application must write to the DIEPCTLn register. Typically, the application must perform a read-modify-write on the DIEPCTLn to avoid modifying the contents of the register beyond setting the Endpoint Enable bit.

3. The application can write all data packets to be transmitted in the next microframe to the FIFO.

The following figure charts this flow.

Figure 38-89. Periodic Data Packet Write in Slave Mode



Interrupt IN Endpoint Behavior in Shared FIFO Operation

In shared FIFO operation, an Interrupt IN Endpoint can be configured to behave as a periodic IN Endpoint, or as a non periodic IN Endpoint, based on the value programmed in the DIEPCTLn.TxFNum field.

- Configured as a non-periodic Endpoint, the application must follow the steps listed under the Non-Periodic IN Endpoint related sections.
- Configured as a periodic IN Endpoint, the application must follow the steps listed under the periodic IN Endpoint related sections.

Setting Global Non-periodic IN Endpoint NAK

Global non-periodic IN Endpoint NAK setting is normally required only in Shared FIFO operation.

Internal Data Flow

1. When the application sets the Global Non-periodic IN NAK bit (DCTL.SGNPInNak), the function stops transmitting data on the non-periodic Endpoint, irrespective of data availability in the Non-periodic Transmit FIFO.
   - Non-isochronous IN tokens receive a NAK handshake reply
2. The function asserts the GINTSTS.GINNakEff interrupt in response to the DCTL.SGNPInNak bit.
3. Once the application detects this interrupt, it can assume that the function is in the Global Non-periodic IN NAK mode. The application can clear this interrupt by clearing the DCTL.SGNPInNak bit.

Application Programming Sequence

1. To stop transmitting any data on non-periodic IN endpoints, the application must set the DCTL.SGNPInNak bit. To set this bit, the following field must be programmed
   - DCTL.SGNPInNak = 1
2. Wait for the assertion of the GINTSTS.GINNakEff interrupt. This interrupt indicates the function has stopped transmitting data on the non-periodic endpoints.
3. The function can transmit valid non-periodic IN data after the application has set the DCTL.SGNPInNak bit, but before the assertion of the GINTSTS.GINNakEff interrupt.
4. The application can optionally mask this interrupt temporarily by writing to the GINTMSK.GINNakEffMsk bit.
   - GINTMSK.GINNakEffMsk = 0
5. To exit Global Non-periodic IN NAK mode, the application must clear the DCTL.SGNPInNak. This also clears the GINTSTS.GINNakEff interrupt.
   - DCTL.SGNPInNak = 1
6. If the application has masked this interrupt earlier, it must be unmasked as follows:
   - GINTMSK.GINNakEffMske = 1

Setting IN Endpoint NAK

Internal Data Flow

1. When the application sets the IN NAK for a particular Endpoint, the function stops transmitting data on the Endpoint, irrespective of data availability in the Endpoint's transmit FIFO.
   - Non-isochronous IN tokens receive a NAK handshake reply
   - Isochronous IN tokens receive a zero-data length packet reply
2. The function asserts the DIEPINTn.IN NAK Effective interrupt in response to the DIEPCTL.Set NAK bit.
3. Once this interrupt is seen by the application, the application can assume that the Endpoint is in IN NAK mode. This interrupt can be cleared by the application by setting the DIEPCTLn. Clear NAK bit.

Application Programming Sequence

1. To stop transmitting any data on a particular IN Endpoint, the application must set the IN NAK bit. To set this bit, the following field must be programmed.
   - DIEPCTLn.SetNAK = 1

2. Wait for assertion of the DIEPINTn.NAK Effective interrupt. This interrupt indicates the function has stopped transmitting data on the Endpoint.
3. The function can transmit valid IN data on the Endpoint after the application has set the NAK bit, but before the assertion of the NAK Effective interrupt.
4. The application can mask this interrupt temporarily by writing to the DIEPMSK.NAK Effective bit.
   - DIEPMSK.NAK Effective = 0
5. To exit Endpoint NAK mode, the application must clear the DIEPCTLn.NAK status. This also clears the DIEPINTn.NAK Effective interrupt.
   - DIEPCTLn.ClearNAK = 1
6. If the application masked this interrupt earlier, it must be unmasked as follows:
   - DIEPMSK.NAK Effective = 1

Disabling a Non-periodic IN Endpoint: Shared FIFO

To disable even a single non-periodic IN Endpoint, the application must disable all non-periodic endpoints. An Endpoint can only be disabled if it is already enabled.

Application Programming Sequence

1. In Slave mode, the application must stop writing data for any non-periodic IN Endpoint on the AHB before disabling any non-periodic IN Endpoint.
2. Before disabling non-periodic IN endpoints, the application must enable the Global IN NP NAK mode in the function. See *Setting Global Non-periodic IN Endpoint NAK* on page 1462.
   - DCTL.Set Global IN NP NAK = 1
3. Wait for the GINTSTS.Global IN NP NAK Effective interrupt.
4. The application must disable all non-periodic endpoints, one after the other, by programming the following fields in the DIEPCTLn register.
   - DIEPCTLn.Endpoint Disable = 1
   - DIEPCTLn.SNAK = 1

   When operating in DMA mode, the function at this point stops fetching data to the Non-periodic FIFO.
5. Asserting the DIEPINTn.Endpoint Disabled interrupt indicates that the function has disabled the Endpoint. This interrupt is asserted for all disabled IN endpoints. Along with asserting this interrupt, the function clears the following bits for all disabled IN endpoints.
   - DIEPCTLn.Endpoint Disable = 0
   - DIEPCTLn.Endpoint Enable = 0
6. The application must read the DIEPTSIZn register for all disabled non-periodic IN endpoints, to calculate how much data on each Endpoint was transmitted on the USB.
7. The application must flush the data in the Non-periodic Transmit FIFO by setting the following fields in the GRSTCTL register.
   - GRSTCTL.TxFIFONum = 0
   - GRSTCTL.TxFFlush = 1
8. The application must poll the GRSTCTL register until the function clears the TxFFlush bit, indicating the end of the flush.

At this point, the application can assume all non-periodic IN endpoints to be disabled.

9. The application can clear the DCTL.Global IN NP NAK bit and, if required, can program the required non-periodic IN endpoints to transmit the new data at a later point.
    - DCTL.Clear Global IN NP NAK = 1
    - GINTMSK.Global IN NP NAK Effective = 1

Periodic IN Endpoint Disable in Shared FIFO Operation

Use the following sequence to disable a specific Periodic IN Endpoint that was previously enabled in shared FIFO operation.

Application Programming Sequence

1. In Slave mode, the application must stop writing data on the AHB to disable the periodic IN Endpoint.
2. The application must set the Endpoint in NAK mode. See *Setting IN Endpoint NAK* on page 1462.
    - DIEPCTLn.SetNAK = 1
3. Wait for the DIEPINTn.NAK Effective interrupt.
4. Set the following bits in the DIEPCTLn register for the Endpoint to be disabled. DIEPCTLn.Endpoint Disable = 1
5. By asserting the DIEPINTn.Endpoint Disabled interrupt, the function indicates that it has completely disabled the specified Endpoint. Additionally, the function clears the following bits.
    - DIEPCTLn.EPEnable = 0
    - DIEPCTLn.EPDisable = 0
6. The application must read the DIEPTSIZn register for the periodic IN Endpoint to calculate how much data on the Endpoint was transmitted on the USB.
7. The application must flush the data in the Periodic Transmit FIFO by setting the following fields in the GRSTCTL register.
    - GRSTCTL.TxFIFONum = Periodic FIFO Number
    - GRSTCTL.TxFFlush = 1
8. The application must poll the GRSTCTL register until the function clears the TxFFlush bit, indicating the end of the flush. The application can re-enable the Endpoint to transmit new data on this Endpoint later.

Generic Non-periodic IN Data Transfers Without Thresholding

This section describes a regular non periodic IN data transfer when transmit thresholding is not enabled.

Application Requirements

1. Before setting up an IN transfer, the application must ensure that all data to be transmitted as part of the IN transfer is part of a single buffer, and must program the size of that buffer and its start address (in DMA mode) to the Endpoint-specific registers.
2. For IN transfers, the Transfer Size field in the Endpoint Transfer Size register denotes a payload that constitutes multiple maximum-packet-size packets and a single short packet. This short packet is transmitted at the end of the transfer.
    - To transmit a few maximum-packet-size packets and a short packet at the end of the transfer:
        – Transfer size[epnum] = n * mps[epnum] + sp (where n is an integer ≥ 0, and $0 \leq sp < mps[epnum]$)
        – If (sp > 0), then packet count[epnum] = n + 1. Otherwise, packet count[epnum] = n

- To transmit a single zero-length data packet:
  - Transfer size[epnum] = 0
  - Packet count[epnum] = 1
- To transmit a few maximum-packet-size packets and a zero-length data packet at the end of the transfer, the application must split the transfer in two parts. The first sends maximum-packetsize data packets and the second sends the zero-length data packet alone.
  - First transfer: transfer size[epnum] = n * mps[epnum]; packet count = n;
  - Second transfer: transfer size[epnum] = 0; packet count = 1;

3. In DMA mode, the function fetches an IN data packet from the memory, always starting at a DWORD boundary. If the maximum packet size of the IN Endpoint is not a multiple of four, the application must arrange the data in the memory with pads inserted at the end of a maximum-packet-size packet so that a new packet always starts on a DWORD boundary.
4. Once an Endpoint is enabled for data transfers, the function updates the Transfer Size register. At the end of IN transfer, which ended with a Timeout (only in Shared FIFO operation) or Endpoint Disabled interrupt, the application must read the Transfer Size register to determine how much data posted in the transmit FIFO was already sent on the USB.
   - Data fetched into transmit FIFO = Application-programmed initial transfer size – function-updated final transfer size
   - Data transmitted on USB = (application-programmed initial packet count – function updated final packet count) * mps[epnum]
   - Data yet to be transmitted on USB = (Application-programmed initial transfer size – data transmitted on USB)
5. In Shared FIFO operation (OTG_EN_DED_TX_FIFO = 0) because all non-periodic IN endpoints share the same transmit FIFO, data for these endpoints must be written to the FIFO in the same Endpoint order in which the USB Host is to access it. The application must therefore be aware of the sequence in which the USB Host is to access the non-periodic IN endpoints. Failure can result in an Endpoint mismatch scenario, if the Endpoint sequence doesn't match. If dedicated FIFO operation is chosen, then there is no such restriction.

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the Endpoint-specific registers and enable the Endpoint to transmit the data.
2. In Slave mode, the application must also write the required data to the transmit FIFO for the Endpoint. In DMA mode, the function fetches the data from memory according to the application setting for the Endpoint.
3. Every time a packet is written into the transmit FIFO by the function's internal DMA (in DMA mode) or the function's application (in Slave mode), the transfer size for that Endpoint is decremented by the packet size. The data is fetched from the memory (DMA/Application), until the transfer size for the Endpoint becomes 0. In Shared FIFO operation (OTG_EN_DED_TX_FIFO = 0), after writing the data into the FIFO, an entry is made in the non-periodic queue. For zero-length packets, there is an entry in the non-periodic queue, without any data in the FIFO.
4. Once the data is written to the transmit FIFO, the function reads it out upon receiving an IN token. For every non-isochronous IN data packet transmitted with an ACK handshake, the packet count for the Endpoint is decremented by one, until the packet count is zero. The packet count is not decremented on a TIMEOUT.
5. For zero-length packets (indicated by zero-length entry in the queue for Shared FIFO operation or by an internal zero-length flag in Dedicated FIFO operation), the function sends out a zero-length packet for the IN token and decrements the Packet Count field.

6. If there is no data in the FIFO for a received IN token and the packet count field for that Endpoint is zero, the function generates a IN Tkn Rcvd When FIFO Empty Interrupt for the Endpoint, provided the Endpoint NAK bit is not set. The function responds with a NAK handshake for non-isochronous endpoints on the USB.
7. In Shared FIFO operation, when there is data for another Endpoint at the top of the Non-periodic Transmit FIFO, the function generates an IN Tkn Rcvd With EPMismatch interrupt for the Endpoint that received the IN token and responds with a NAK handshake for the non-ISO endpoints on the USB. This interrupt is not generated in Dedicated FIFO operation.
8. In Shared FIFO operation, if there is a TIMEOUT condition, the DIEPCTLn.Timeout interrupt is generated. In Dedicated FIFO operation, the function internally rewinds the FIFO pointers and no timeout interrupt is generated.
9. When the transfer size is 0 and the packet count is 0, the transfer complete interrupt for the Endpoint is generated and the Endpoint enable is cleared.

Application Programming Sequence

1. Program the DIEPTSIZn register with the transfer size and corresponding packet count. In DMA mode, also program the DIEPDMAn register.
2. When using Shared FIFO operation in Slave mode, when transmitting a zero-length data packet in Slave mode, check the GNPTXSTS register to ensure that at least one space is available in the Non-periodic Request Queue (NPTxQSpcAvail). No space is required in the Data FIFO(NPTxFSpcAvail) is required to transmit a zero-data length packet. This checking is not required in Dedicated FIFO operation.
3. Program the DIEPCTLn register with the Endpoint characteristics and set the CNAK and Endpoint Enable bits. In DMA mode, ensure that the NextEp field is programmed so that the function fetches the data for IN endpoints in the correct order. See *Non-periodic IN Endpoint Sequencing* on page 1489 for details.
4. When using Shared FIFO operation, in Slave mode, when transmitting a non-zero length data packet, wait for the GINTSTS.NonPeriodic TxF Empty interrupt and write the data to the Non-periodic Transmit FIFO as described in *Non-periodic Packet Write in Slave Mode: Shared FIFO* on page 1459.

   This step can be repeated multiple times, depending on the transfer size.
5. When using dedicated FIFO operation in slave mode, when transmitting non-zero length data packet, the application must poll the DTXFSTSn register (where n is the FIFO number associated with that Endpoint) to determine whether there is enough space in the data FIFO. The application can optionally use DIEPINTn.TxFEmp before writing the data.
6. Assertion of the DIEPINTn.TimeOut interrupt indicates that a timeout condition is detected on this Endpoint. See *Timeout for Non-periodic IN Data Transfers: Shared FIFO* on page 1472.
7. Assertion of DIEPINTn.TimeOut Interrupt in Shared FIFO operation, indicates that a TIMEOUT handshake is seen on this Endpoint. Refer to the Section TimeOUT for non-periodic IN Data Transfers. DIEPINTn.TimeOut Interrupt is not generated in Dedicated FIFO operation and the function handles this internally.

Generic Non-periodic IN Data Transfers With Thresholding

This section describes a regular non periodic IN data transfer when transmit thresholding is enabled.

Application Requirements

Application requirements are the same as those for DMA mode with no thresholding.

Internal Data Flow

1. The application should set the transfer size and packet count fields in the Endpoint Specific registers, and enable the Endpoint to transmit the data.

2. The function fetches threshold amount of data for one Endpoint before switching to the next in a round robin fashion with equal fairness. The priority is given to periodic endpoints. non-periodic Endpoint data is fetched only if data for the periodic endpoints has been fetched or if the currently active token on the USB is non-periodic. The function does not switch, and continue to fetch till the complete packet, if it finds that the currently active IN token on the USB is for that particular Endpoint and the FIFO does not have the complete packet.
3. In response to an IN token on the USB, the function starts transmitting data, if the MAC finds at least a threshold amount of data in the FIFO for that particular Endpoint.
4. With each threshold amount of data written into the FIFO, the transfer size for that Endpoint is decremented by the threshold size, except for the last packet. For the last packet, the transfer size is not decremented by the function. After writing the first threshold amount of data into the FIFO, the "number of packets in FIFO" count is incremented. For zero-length packets, a separate flag is set for that Endpoint FIFO, without any data in the FIFO. This count is internally maintained by the function and is decremented when a full packet has been read out of the FIFO.
5. If the MAC sees an under run case, where there is not enough data in the FIFO, the function corrupts the data (invert the CRC) on the USB. The function internally flushes the FIFO, rewinding the DMA pointers and refetchs the packet(s). Also the DIEPINTn.TxfifoUndrn bit is set as an indication to the application.
6. For every non-ISO data IN packet transmitted, with an ACK handshake, the packet count for the Endpoint is decremented by one, until the packet count becomes zero. The packet count is not decremented on a TIMEOUT or under run condition.
7. If the zero length flag in the FIFO is set (internally set by the function, when the application enables and Endpoint for zero length), then function sends out zero length packet for the IN token and decrements the packet count field.
8. If there is no data (or partial threshold data) in the FIFO for a received IN token, and the packet count field for that Endpoint is 0, the function generates a IN Tkn Rcvd When FIFO Empty Interrupt for the Endpoint. The function responds with a NAK handshake for the non-ISO endpoints on the USB.
9. If the function does not receive a handshake after sending the packet (TIMEOUT), the function internally flushes the FIFO, rewinds the DMA pointers and refetchs the packet(s).
10. When the packet count is zero, the transfer complete interrupt for the Endpoint is generated, and the Endpoint enable and transfer size are both cleared by the function.

Application Programming Sequence

1. Program the DIEPTSIZn register, for the transfer size and the corresponding packet count and the DIEPDMAn register.
2. Program the DIEPCTLn register, with the Endpoint Characteristics and set the CNAK and the Endpoint Enable bit. Also specify the Tx FIFO number in the DIEPCTLn.TXFNum field.
3. Assertion of DIEPINTn.XferCompl interrupt marks the successful completion of the non-periodic IN transfer. A read to the DIEPTSIZn register should indicate a transfer size = 0 and packet count = 0, indicating all the data is transmitted on the USB.

Generic Periodic IN Data Transfers Without Thresholding

This section describes a typical Periodic IN data transfer when thresholding is not enabled.

Application Requirements

1. Application requirements 1, 2, 3, and 4 of *Generic Periodic IN Data Transfers Without Thresholding* on page 1467 also apply to periodic IN data transfers, except for a slight modification of Requirement 2.

- The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum-packet-size packets and a short packet at the end of the transfer, the following conditions must be met:
  - transfer size[epnum] = n * mps[epnum] + sp (where n is an integer ≥ 0, and 0 ≤ sp < mps[epnum])
  - If (sp > 0), packet count[epnum] = n + 1 Otherwise, packet count[epnum] = n;
  - mc[epnum] = packet count[epnum]
- The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by itself. To transmit a single zero-length data packet,
  - transfer size[epnum] = 0
  - packet count[epnum] = 1
  - mc[epnum] = packet count[epnum]

2. The application can only schedule data transfers 1 microframe at a time.
   - (DIEPTSIZn.MC – 1) * DIEPCTLn.MPS ≤ DIEPTSIZn.XferSiz ≤ DIEPTSIZn.MC * DIEPCTLn.MPS
   - DIEPTSIZn. PktCnt = DIEPTSIZn.MC
   - If DIEPTSIZn.XferSiz < DIEPTSIZn.MC * DIEPCTLn.MPS, the last data packet of the transfer is a short packet.
3. The application can schedule data transfers for multiple microframes, only if multiples of max packet sizes (up to 3 packets), should be transmitted every microframe. This is can be done, only when the function is operating in DMA mode. This is not a recommended mode of operation though.
   - ((n*DIEPTSIZn.MC) – 1)*DIEPCTLn.MPS ≤ DIEPTSIZn.Transfer Size ≤ n*DIEPT– SIZn.MC*DIEPCTLn.MPS
   - DIEPTSIZn.Packet Count = n*DIEPTSIZn.MC
   - n is the number of microframes for which the data transfers are scheduled

   Data Transmitted per microframe in this case would be DIEPTSIZn.MC*DIEPCTLn.MPS, in all the microframes except the last one. In the microframe n, the data transmitted would be (DIEPTSIZn.TransferSize – (n – 1)*DIEPTSIZn.MC*DIEPCTLn.MPS)
4. For Periodic IN endpoints, the data must always be prefetched 1 microframe ahead for transmission in the next microframe. This can be done, by enabling the Periodic IN Endpoint 1 (micro)frame ahead of the (micro)frame in which the data transfer is scheduled.
5. The complete data to be transmitted in the (micro)frame must be written into the transmit FIFO (either by the application or the DMA), before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per microframe is missing in the transmit FIFO when the Periodic IN token is received, the function behaves as when the FIFO was empty. When the transmit FIFO is empty,
   - A zero-data length packet would be transmitted on the USB for ISO IN endpoints
   - A NAK handshake would be transmitted on the USB for INTR IN endpoints

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the Endpoint-specific registers and enable the Endpoint to transmit the data.
2. In Slave mode, the application must also write the required data to the associated transmit FIFO for the Endpoint. In DMA mode, the function fetches the data for the Endpoint from memory, according to the application setting.

3. Every time either the function's internal DMA (in DMA mode) or the function's the application writes a packet to the transmit FIFO, the transfer size for that Endpoint is decremented by the packet size. The data is fetched from the DMA or application memory until the transfer size for the Endpoint becomes 0.
4. When an IN token is received for an periodic Endpoint, the function transmits the data in the FIFO, if available. If the complete data payload (complete packet, in dedicated FIFO mode) for the microframe is not present in the FIFO, then the function generates an IN Tkn Rcvd When TxF Empty Interrupt for the Endpoint.
   - A zero-length data packet is transmitted on the USB for isochronous IN endpoints
   - A NAK handshake is transmitted on the USB for interrupt IN endpoints
5. The packet count for the Endpoint is decremented by 1 under the following conditions:
   - For isochronous endpoints, when a zero- or non-zero length data packet is transmitted
   - For interrupt endpoints, when an ACK handshake is transmitted
6. When the transfer size and packet count are both 0, the Transfer Completed interrupt for the Endpoint is generated and the Endpoint enable is cleared.
7. At the "Periodic frame Interval" (controlled by DCFG.PerFrint), when the function finds non-empty any of the isochronous IN Endpoint FIFOs scheduled for the current (micro)frame non-empty, the function generates a GINTSTS.incompISOIN interrupt.

Application Programming Sequence (Transfer Per Microframe)

1. Program the DIEPTSIZn register. In DMA mode, also program the DIEPDMAn register.
2. Program the DIEPCTLn register with the Endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. In Slave mode, write the data to be transmitted in the next microframe to the transmit FIFO as described in *Periodic Packet Write in Slave Mode: Shared FIFO* on page 1460.
4. Asserting the DIEPINTn.In Token Rcvd When TxF Empty interrupt indicates that either the DMA or the application has not yet written all data to be transmitted to the transmit FIFO.
   - If the interrupt Endpoint is already enabled when this interrupt is detected, ignore the interrupt. If it is not enabled, enable the Endpoint so that the data can be transmitted on the next IN token attempt.
   - If the isochronous Endpoint is already enabled when this interrupt is detected, see *Incomplete Isochronous IN Data Transfers* on page 1471 for more details.
5. The function handles timeouts internally on interrupt IN endpoints programmed as periodic endpoints, or when Dedicated FIFO operation is used, without application intervention. The application, thus, never detects a DIEPINTn.TimeOUT interrupt for periodic interrupt IN endpoints.
6. Asserting the DIEPINTn.XferCompl interrupt with no DIEPINTn.In Tkn Rcvd When TxF Empty interrupt indicates the successful completion of an isochronous IN transfer. A read to the DIEPTSIZn register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
7. Asserting the DIEPINTn.XferCompl interrupt, with or without the DIEPINTn.In Tkn Rcvd When TxF Empty interrupt, indicates the successful completion of an interrupt IN transfer. A read to the DIEPTSIZn register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
8. Asserting the GINTSTS.incomplete Isochronous IN Transfer interrupt with none of the aforementioned interrupts indicates the function did not receive at least 1 periodic IN token in the current microframe.

   For isochronous IN endpoints, see *Incomplete Isochronous IN Data Transfers* on page 1471, for more details.

Generic Periodic IN Data Transfers With Thresholding

This section describes a typical Periodic IN data transfer when transmit thresholding is enabled.

Application Requirements

Application requirements are the same as those for DMA mode with no thresholding.

1. Application requirements 1, 2, 3, and 4 of *Generic Periodic IN Data Transfers With Thresholding* on page 1470 also apply to periodic IN data transfers, except for a slight modification of Requirement 2.
   - The application can only transmit multiples of maximum-packet-size data packets or multiples of maximum-packet-size packets, plus a short packet at the end. To transmit a few maximum-packet-size packets and a short packet at the end of the transfer, the following conditions must be met.
     – transfer size[epnum] = n * mps[epnum] + sp (where n is an integer ≥ 0, and $0 \leq sp < mps[epnum]$)
     – If ($sp > 0$), packet count[epnum] = n + 1 Otherwise, packet count[epnum] = n;
     – mc[epnum] = packet count[epnum]
   - The application cannot transmit a zero-length data packet at the end of transfer. It can transmit a single zero-length data packet by it self. To transmit a single zero-length data packet,
     – transfer size[epnum] = 0
     – packet count[epnum] = 1
     – mc[epnum] = packet count[epnum]
2. The application can only schedule data transfers 1 microframe at a time.
   - (DIEPTSIZn.MC – 1) * DIEPCTLn.MPS $\leq$ DIEPTSIZn.XferSiz $\leq$ DIEPTSIZn.MC * DIEPCTLn.MPS
   - DIEPTSIZn. PktCnt = DIEPTSIZn.MC
   - If DIEPTSIZn.XferSiz $<$ DIEPTSIZn.MC * DIEPCTLn.MPS, the last data packet of the transfer is a short packet.
3. For Periodic IN endpoints, the data must always be prefetched 1 microframe ahead for transmission in the next microframe. This can be done, by enabling the Periodic IN Endpoint 1 (micro)frame ahead of the (micro)frame in which the data transfer is scheduled.
4. The complete data to be transmitted in the (micro)frame must be written into the transmit FIFO by the application, before the Periodic IN token is received. Even when 1 DWORD of the data to be transmitted per microframe is missing in the transmit FIFO when the Periodic IN token is received, the function behaves as when the FIFO was empty. When the transmit FIFO is empty,
   - A zero-data length packet would be transmitted on the USB for ISO IN endpoints
   - A NAK handshake would be transmitted on the USB for INTR IN endpoints

Internal Data Flow

1. The application must set the Transfer Size and Packet Count fields in the Endpoint-specific registers and enable the Endpoint to transmit the data.
2. In Slave mode, the application must also write the required data to the associated transmit FIFO for the Endpoint.
3. Every time either the function's the application writes a packet to the transmit FIFO, the transfer size for that Endpoint is decremented by the packet size. The data is fetched from the application memory until the transfer size for the Endpoint becomes 0.
4. When an IN token is received for an periodic Endpoint, the function transmits the data in the FIFO, if available. If the complete data payload (complete packet, in dedicated FIFO mode) for the microframe is not present in the FIFO, then the function generates an IN Tkn Rcvd When TxF Empty Interrupt for the Endpoint.

- A zero-length data packet is transmitted on the USB for isochronous IN endpoints
- A NAK handshake is transmitted on the USB for interrupt IN endpoints

5. The packet count for the Endpoint is decremented by 1 under the following conditions:
   - For isochronous endpoints, when a zero- or non-zero length data packet is transmitted
   - For interrupt endpoints, when an ACK handshake is transmitted
6. When the transfer size and packet count are both 0, the Transfer Completed interrupt for the Endpoint is generated and the Endpoint enable is cleared.
7. At the "Periodic frame Interval" (controlled by DCFG.PerFrint), when the function finds non-empty any of the isochronous IN Endpoint FIFOs scheduled for the current (micro)frame non-empty, the function generates a GINTSTS.incompISOIN interrupt.

Application Programming Sequence (Transfer Per Microframe)

1. Program the DIEPTSIZn register, and, in addition, in DMA mode program the DIEPDMAn register..
2. Program the DIEPCTLn register with the Endpoint characteristics and set the CNAK and Endpoint Enable bits.
3. In Slave mode, write the data to be transmitted in the next microframe to the transmit FIFO as described in *Periodic Packet Write in Slave Mode: Shared FIFO* on page 1460.
4. Asserting the DIEPINTn.In Token Rcvd When TxF Empty interrupt indicates that the application has not yet written all data to be transmitted to the transmit FIFO.
   - If the interrupt Endpoint is already enabled when this interrupt is detected, ignore the interrupt. If it is not enabled, enable the Endpoint so that the data can be transmitted on the next IN token attempt.
   - If the isochronous Endpoint is already enabled when this interrupt is detected, see *Incomplete Isochronous IN Data Transfers* on page 1471 for more details.
5. The function handles timeouts internally on interrupt IN endpoints programmed as periodic endpoints, or when Dedicated FIFO operation is used, without application intervention. The application, thus, never detects a DIEPINTn.TimeOUT interrupt for periodic interrupt IN endpoints.
6. Asserting the DIEPINTn.XferCompl interrupt with no DIEPINTn.In Tkn Rcvd When TxF Empty interrupt indicates the successful completion of an isochronous IN transfer. A read to the DIEPTSIZn register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
7. Asserting the DIEPINTn.XferCompl interrupt, with or without the DIEPINTn.In Tkn Rcvd When TxF Empty interrupt, indicates the successful completion of an interrupt IN transfer. A read to the DIEPTSIZn register must indicate transfer size = 0 and packet count = 0, indicating all data is transmitted on the USB.
8. Asserting the GINTSTS.incomplete Isochronous IN Transfer interrupt with none of the aforementioned interrupts indicates the function did not receive at least 1 periodic IN token in the current microframe.
   - For isochronous IN endpoints, see *Incomplete Isochronous IN Data Transfers* on page 1471, for more details.

Incomplete Isochronous IN Data Transfers

This section describes what the application must do on an incomplete isochronous IN data transfer.

Internal Data Flow

1. An isochronous IN transfer is treated as incomplete in one of the following conditions.
   - The function receives a corrupted isochronous IN token on at least one isochronous IN Endpoint. In this case, the application detects a GINTSTS.incomplete Isochronous IN Transfer interrupt.

- The application or DMA is slow to write the complete data payload to the transmit FIFO and an IN token is received before the complete data payload is written to the FIFO. In this case, the application detects a DIEPINTn.IN Tkn Rcvd When TxFIFO Empty interrupt. The application can ignore this interrupt, as it eventually results in a GINTSTS.incomplete Isochronous IN Transfer interrupt at the end of periodic frame.
  - The function transmits a zero-length data packet on the USB in response to the received IN token.
- If thresholding is enabled and there was an under run condition, the function also generates a DINEPINTn.TxfifoUndrn interrupt. The application can ignore this interrupt, which eventually results in a GINTSTS.incomplete ISO IN Transfer interrupt.

2. In any of the aforementioned cases, in Slave mode, the application must stop writing the data payload to the transmit FIFO as soon as possible.
3. The application must set the NAK bit and the disable bit for the Endpoint. In DMA mode, the function automatically stops fetching the data payload when the Endpoint disable bit is set.
4. The function disables the Endpoint, clears the disable bit, and asserts the Endpoint Disable interrupt for the Endpoint.

Application Programming Sequence

1. The application can ignore the DIEPINTn.IN Tkn Rcvd When TxFIFO empty interrupt on any isochronous IN Endpoint, as it eventually results in a GINTSTS.incomplete Isochronous IN Transfer interrupt. The application can also ignore the DIEPINTn.TxfifoUndrn interrupt when thresholding is enabled.
2. Assertion of the GINTSTS.incomplete Isochronous IN Transfer interrupt indicates an incomplete isochronous IN transfer on at least one of the isochronous IN endpoints.
3. The application must read the Endpoint Control register for all isochronous IN endpoints to detect endpoints with incomplete IN data transfers.
4. In Slave mode, the application must stop writing data to the Periodic Transmit FIFOs associated with these endpoints on the AHB.
5. In both modes of operation, program the following fields in the DIEPCTLn register to disable the Endpoint. See *Disabling a Non-periodic IN Endpoint: Shared FIFO* on page 1463 for more details.
   - DIEPCTLn.SetNAK = 1
   - DIEPCTLn.Endpoint Disable = 1
6. The DIEPINTn.Endpoint Disabled interrupt's assertion indicates that the function has disabled the Endpoint.
   - At this point, the application must flush the data in the associated transmit FIFO or overwrite the existing data in the FIFO by enabling the Endpoint for a new transfer in the next microframe. To flush the data, the application must use the GRSTCTL register.

Timeout for Non-periodic IN Data Transfers: Shared FIFO

This section describes how an application must perform in shared FIFO operation when a non-periodic IN Endpoint receives a TIMEOUT handshake from the USB Host.

This section is applicable only for Shared FIFO operation. For Dedicated FIFO operation, timeout is handled internally by the function and the application is not interrupted.

Internal Data Flow

1. When a timeout is detected on any non-periodic IN Endpoint, the function sets the Global NP IN NAK bit internally and NAKs any non-periodic endpoints' IN tokens.
2. The application must disable all non-periodic IN endpoints.

Application Programming Sequence

1. Assertion of the DIEPINTn.TimeOut interrupt indicates that a timeout condition was detected for this IN Endpoint.
2. The application must disable all non-periodic endpoints as indicated in *Disabling a Non-periodic IN Endpoint: Shared FIFO* on page 1463.
3. Assertion of the DIEPINTn.Endpoint Disabled interrupt indicates that the function has disabled the Endpoint.
4. To retransmit the data on the USB from where it stopped before the Endpoint was disabled, the application must flush the Non-periodic Transmit FIFO and re-enable the endpoints.
5. On the Endpoint that received the Timeout Condition interrupt, the application must enable the function to retransmit the data packet that received the Timeout Condition interrupt.

Special Case: Timeout for Control IN Endpoints

The application must treat the timeout interrupt received for the last IN transaction of a control transfer's Data stage differently from a normal timeout, because extra handling is required. This is done to take into account the timeout due to a lost ACK (USB did not receive the ACK send by the Host).

After receiving the timeout interrupt, and making sure that it's for the last packet, the application must perform steps 1 to 5 above, then enable both IN and OUT endpoints of the control Endpoint.

If the timeout is due to a lost ACK, the Host switches to the Data stage, and the application receives Transfer Complete interrupt for the OUT Endpoint. The application can then flush the IN packet and disable both the IN and OUT endpoints. If the timeout was due to lost data, the Host sends the IN token again, and the application receives a Transfer Complete interrupt for the IN Endpoint. The application can thus keep the OUT Endpoint enabled for the status phase.

Stalling Non-Isochronous IN Endpoints

This section describes how the application can stall a non-isochronous Endpoint.

Application Programming Sequence

1. Disable the IN Endpoint to be stalled. See *Disabling a Non-periodic IN Endpoint: Shared FIFO* on page 1463 for more details. Set the Stall bit as well.
   - DIEPCTLn.Endpoint Disable = 1, when the Endpoint is already enabled
   - DIEPCTLn.STALL = 1
   - The Stall bit always takes precedence over the NAK bit
2. Assertion of the DIEPINTn.Endpoint Disabled interrupt indicates to the application that the function has disabled the specified Endpoint.
3. The application must flush the Non-periodic or Periodic Transmit FIFO, depending on the Endpoint type. In case of a non-periodic Endpoint, the application must re-enable the other non-periodic endpoints, which do not need to be stalled, to transmit data.
4. Whenever the application is ready to end the STALL handshake for the Endpoint, the DIEPCTLn.STALL bit must be cleared.
5. If the application sets or clears a STALL for an Endpoint due to a SetFeature.Endpoint Halt command or ClearFeature.Endpoint Halt command, the Stall bit must be set or cleared before the application sets up the Status stage transfer on the control Endpoint.

Special Case: Stalling the Control OUT Endpoint

The function must stall IN/OUT tokens if, during the Data stage of a control transfer, the Host sends more IN/OUT tokens than are specified in the SETUP packet. In this case, the application must to enable DIEPINTn.INTknTXFEmp and DOEPINTn.OUTTknEPdis interrupts during the Data stage of the control transfer, after the function has transferred the amount of data specified in the SETUP packet. Then, when the application receives this interrupt, it must set the STALL bit in the corresponding Endpoint control register, and clear this interrupt.

Examples:

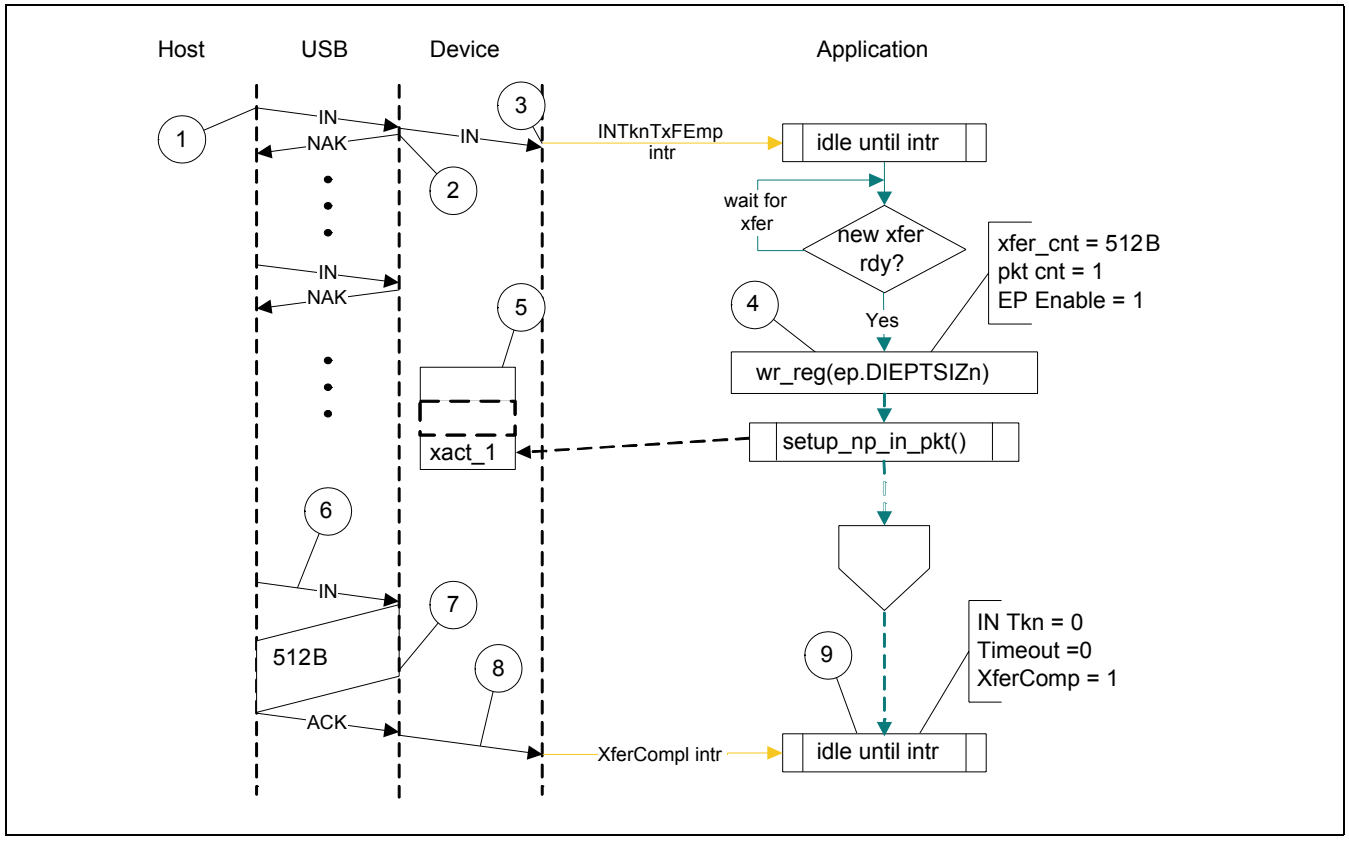
- *Slave Mode Bulk IN Transaction* on page 1474
- *Slave Mode Bulk IN Transfer (Pipelined Transaction)* on page 1475
- *Slave Mode Bulk IN Two-Endpoint Transfer* on page 1476
- *Bulk IN TIMEOUT (Shared FIFO Operation)* on page 1478
- *Bulk IN Stall* on page 1479
- *Bulk IN DMA mode With Thresholding* on page 1481
- *Isochronous IN DMA Mode With Thresholding* on page 1483

Slave Mode Bulk IN Transaction

These notes refer to *Figure 38-90* on page 1475.

1. The Host attempts to read data (IN token) from an Endpoint.
2. On receiving the IN token on the USB, the function returns a NAK handshake, because no data is available in the transmit FIFO.
3. To indicate to the application that there was no data to send, the function generates a DIEPINTn.IN Token Rcvd When TxFIFO Empty interrupt.
4. When data is ready, the application sets up the DIEPTSIZn register with the Transfer Size and Packet Count fields.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The Host reattempts the IN token.
7. Because data is now ready in the FIFO, the function now responds with the data and the Host ACKs it.
8. Because the XferSize is now zero, the intended transfer is complete. The Device function generates a DIEPINTn.XferCompl interrupt.
9. The application processes the interrupt and uses the setting of the DIEPINTn.XferCompl interrupt bit to determine that the intended transfer is complete.

Figure 38-90. Slave Mode Bulk IN Transaction



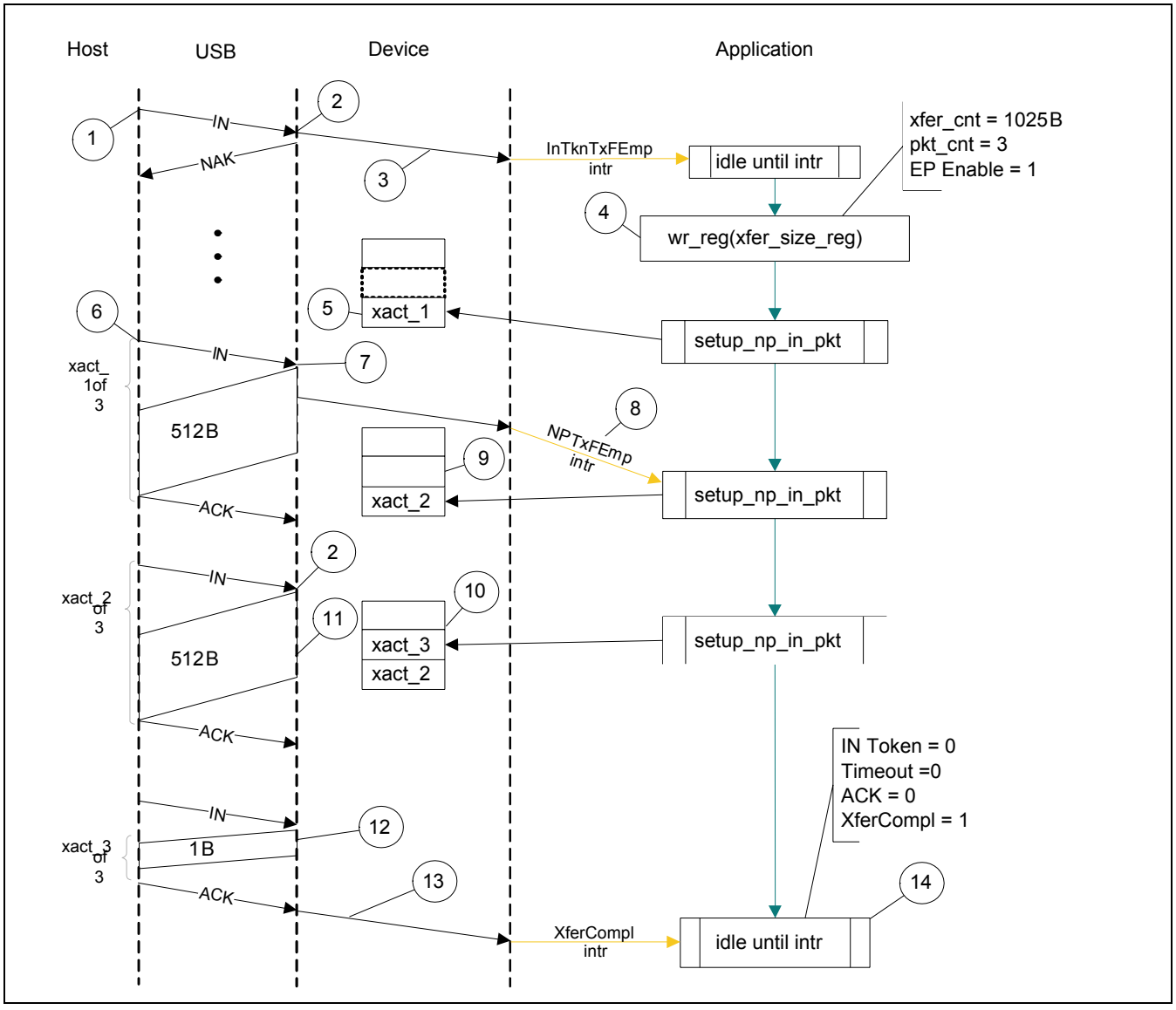
Slave Mode Bulk IN Transfer (Pipelined Transaction)

These notes refer to *Figure 38-91* on page 1476.

1. The Host attempts to read data (IN token) from an Endpoint.
2. On receiving the IN token on the USB, the function returns a NAK handshake, because no data is available in the transmit FIFO.
3. To indicate that there was no data to send, the function generates an DIEPINTn.InTkn Rcvd When TxFIFO Empty interrupt.
4. When data is ready, the application sets up the DIEPTSIZn register with the transfer size and packet count.
5. The application writes one maximum packet size or less of data to the Non-periodic TxFIFO.
6. The Host reattempts the IN token.
7. Because data is now ready in the FIFO, the function responds with the data, and the Host ACKs it.
8. When the TxFIFO level falls below the halfway mark, the function generates a GINTSTS.NonPeriodic TxFIFO Empty interrupt. This triggers the application to start writing additional data packets to the FIFO.
9. A data packet for the second transaction is ready in the TxFIFO.
10. A data packet for third transaction is ready in the TxFIFO while the data for the second packet is being sent on the bus.
11. The second data packet is sent to the Host.
12. The last short packet is sent to the Host.

13. Because the last packet is sent and XferSize is now zero, the intended transfer is complete. The function generates a DIEPINTn.XferCompl interrupt.
14. The application processes the interrupt and uses the setting of the DIEPINTn.XferCompl interrupt bit to determine that the intended transfer is complete.

Figure 38-91. Slave Mode Bulk IN Transfer (Pipelined Transaction)



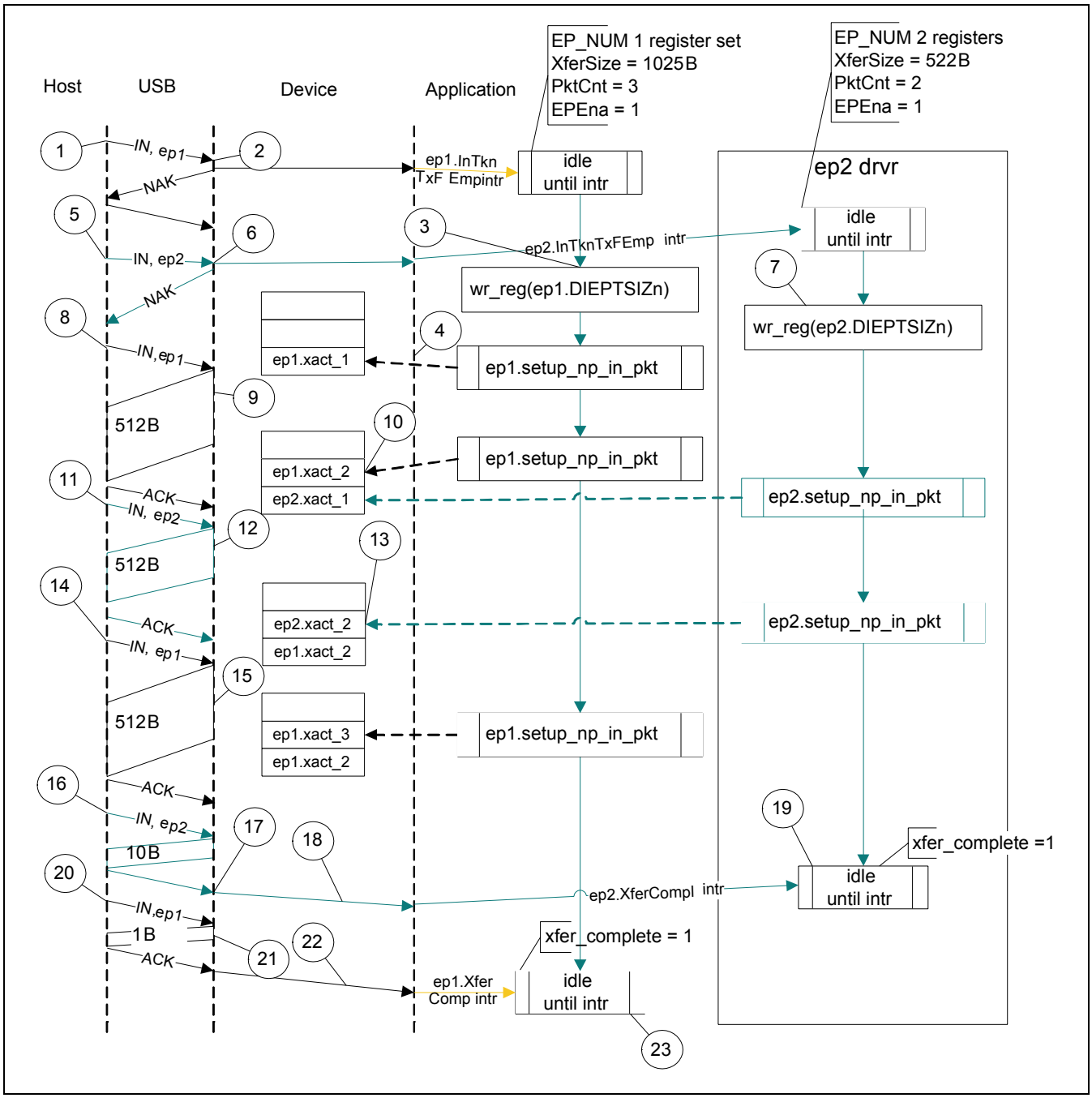
Slave Mode Bulk IN Two-Endpoint Transfer

These notes refer to *Figure 38-92* on page 1478.

1. The Host attempts to read data (IN token) from Endpoint 1.
2. On receiving the IN token on the USB, the function returns a NAK handshake, because no data is available in the transmit FIFO for Endpoint 1, and generates a DIEPINT1.InTkn Rcvd When TxFIFO Empty interrupt.
3. The application processes the interrupt and initializes DIEPTSIZ1 register with the Transfer Size and Packet Count fields. The application starts writing the transaction data to the transmit FIFO.

4. The application writes one maximum packet size or less of data for Endpoint 1 to the Non-periodic TxFIFO.
5. Meanwhile, the Host attempts to read data (IN token) from Endpoint 2.
6. On receiving the IN token on the USB, the function returns a NAK handshake, because no data is available in the transmit FIFO for Endpoint 2, and the function generates a DIEPINT2.InTkn Rcvd When TxFIFO Empty interrupt.
7. Because the application has completed writing the packet for Endpoint 1, it initializes the DIEPTSIZ2 register with the Transfer Size and Packet Count fields. The application starts writing the transaction data into the transmit FIFO for Endpoint 2.
8. The Host repeats its attempt to read data (IN token) from Endpoint 1.
9. Because data is now ready in the TxFIFO, the function returns the data, which the Host ACKs.
10. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact1 and ep1.xact2, in order).
11. The Host repeats its attempt to read data (IN token) from Endpoint 2.
12. Because Endpoint 2's data is ready, the function responds with the data (ep2.xact_1), which the Host ACKs.
13. Meanwhile, the application has initialized the data for the next two packets in the TxFIFO (ep2.xact2 and ep1.xact3, in order). The application has finished initializing data for the two endpoints involved in this scenario.
14. The Host repeats its attempt to read data (IN token) from Endpoint 1.
15. Because data is now ready in the FIFO, the function responds with the data, which the Host ACKs.
16. The Host repeats its attempt to read data (IN token) from Endpoint 2.
17. With data now ready in the FIFO, the function responds with the data, which the Host ACKs.
18. With the last packet for Endpoint 2 sent and its XferSize now zero, the intended transfer is complete. The function generates a DIEPINT2.XferCompl interrupt for this Endpoint.
19. The application processes the interrupt and uses the setting of the DIEPINT2.XferCompl interrupt bit to determine that the intended transfer on Endpoint 2 is complete.
20. The Host repeats its attempt to read data (IN token) from Endpoint 1 (last transaction).
21. With data now ready in the FIFO, the function responds with the data, which the Host ACKs.
22. Because the last Endpoint one packet has been sent and XferSize is now zero, the intended transfer is complete. The function generates a DIEPINT1.XferCompl interrupt for this Endpoint.
23. The application processes the interrupt and uses the setting of the DIEPINT1.XferCompl interrupt bit to determine that the intended transfer on Endpoint 1 is complete.

Figure 38-92. Slave Mode Bulk IN Two-Endpoint Transfer



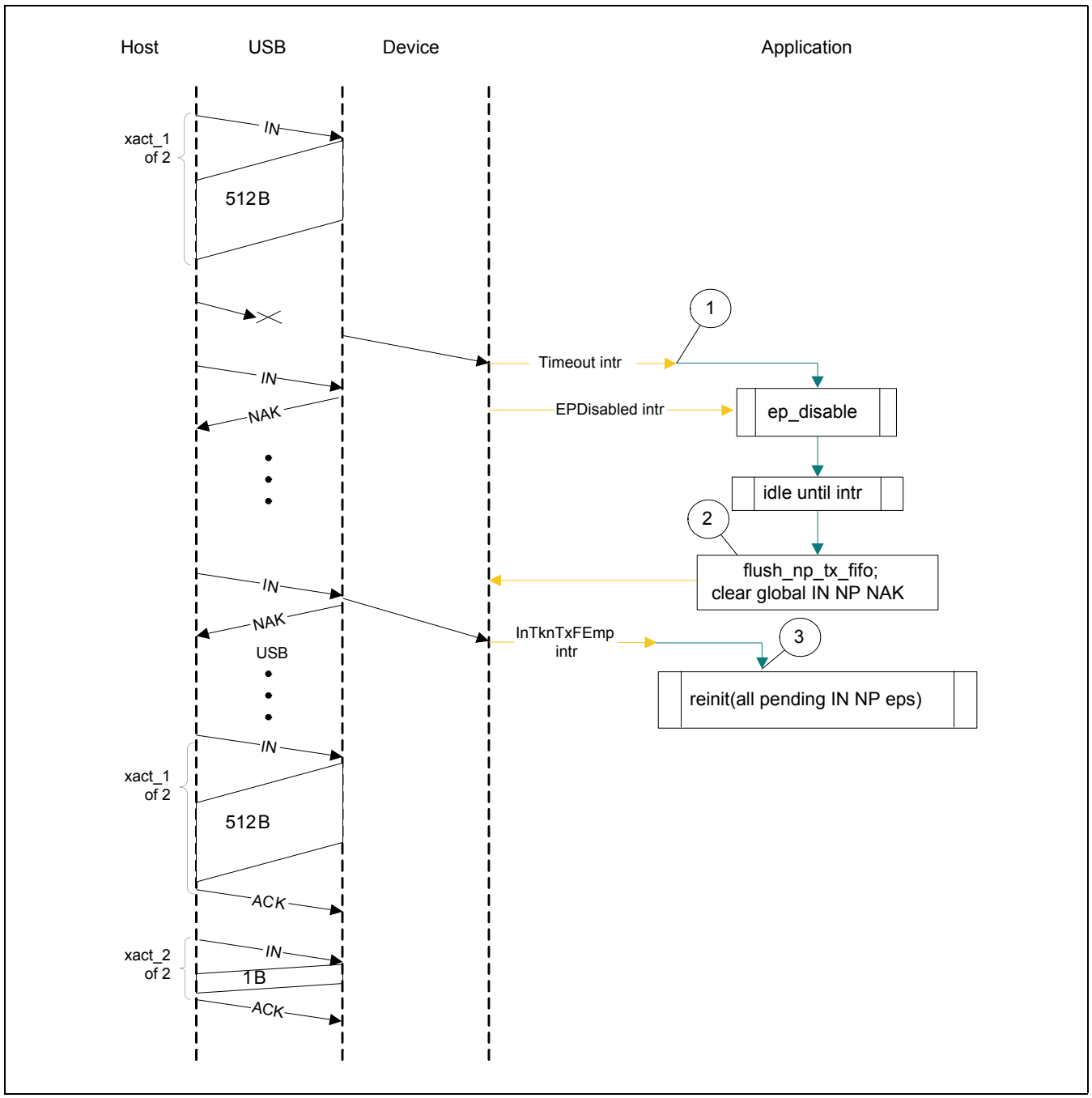
Bulk IN TIMEOUT (Shared FIFO Operation)

These notes refer to *Figure 38-93* on page 1479.

1. The application has detected a DIEPINTn.TimeOUT interrupt on a particular Endpoint. The application must disable the Endpoint and wait for the DIEPINTn.EndpointDisabled interrupt.
2. After the Endpoint is disabled, the application must flush the Non-periodic Transmit FIFO and set DCTL.CGNPInNak.

3. When a DIEPINTn.INTknTXFEmp interrupt, the application must reschedule the remaining part of the transfer to be transmitted on the USB.

Figure 38-93. Bulk IN TIMEOUT (Shared FIFO Operation)



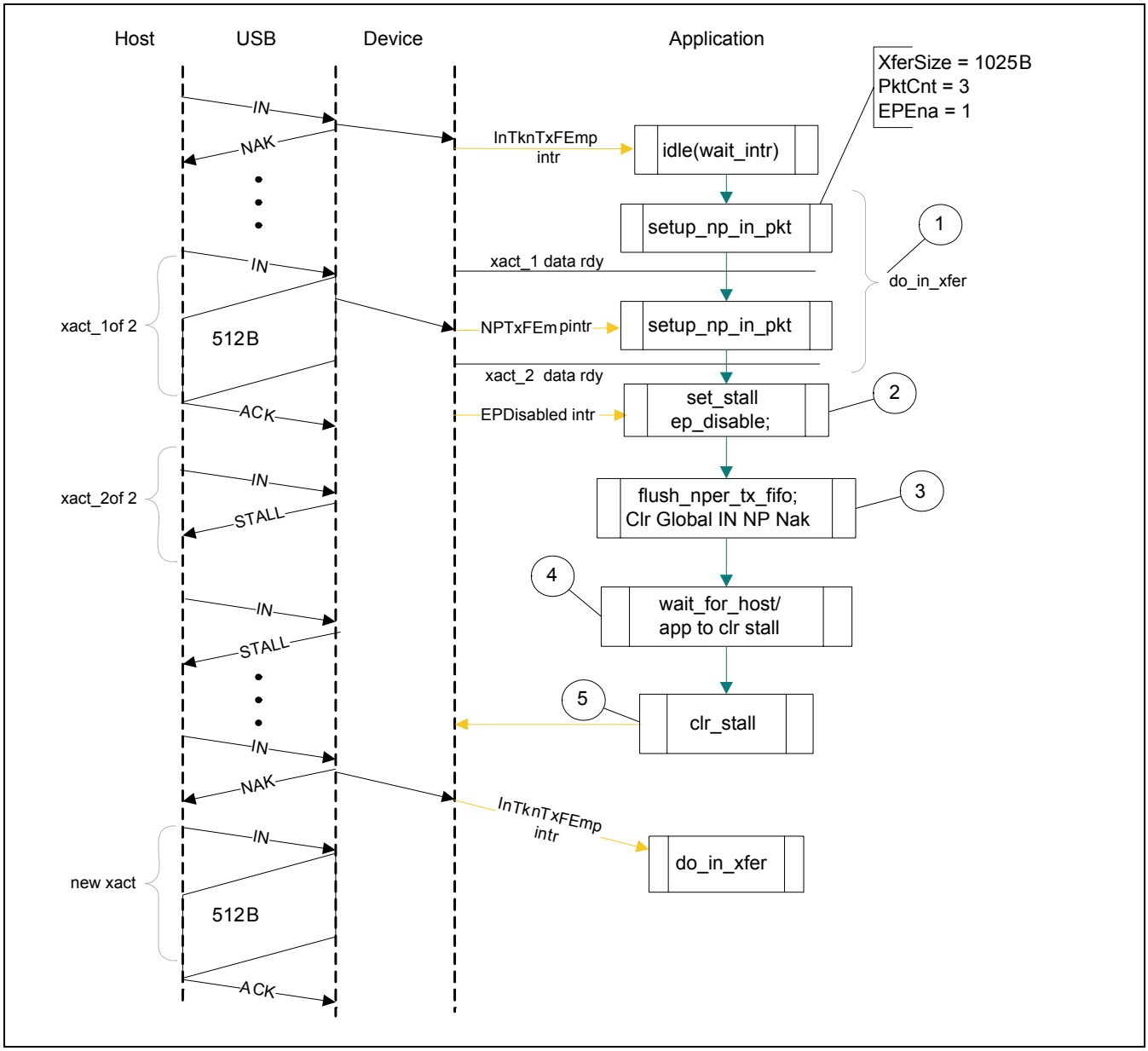
Bulk IN Stall

These notes refer to *Figure 38-94* on page 1480.

1. The application has scheduled an IN transfer on receiving the DIEPINTn.InTknRcvd When TxFIFO Empty interrupt.

2. When the transfer is in progress, the application must force a STALL on the Endpoint. This could be because the application has received a SetFeature.Endpoint Halt command. The application sets the Stall bit, disables the Endpoint and waits for the DIEPINTn.Endpoint Disabled interrupt. This generates STALL handshakes for the Endpoint on the USB.
3. On receiving the interrupt, the application flushes the Non-periodic Transmit FIFO and clears the DCTL.GlobalINNPNAK bit.
4. On receiving the ClearFeature.Endpoint Halt command, the application clears the Stall bit.
5. The Endpoint behaves normally and the application can re-enable the Endpoint for new transfers.

Figure 38-94. Bulk IN Stall

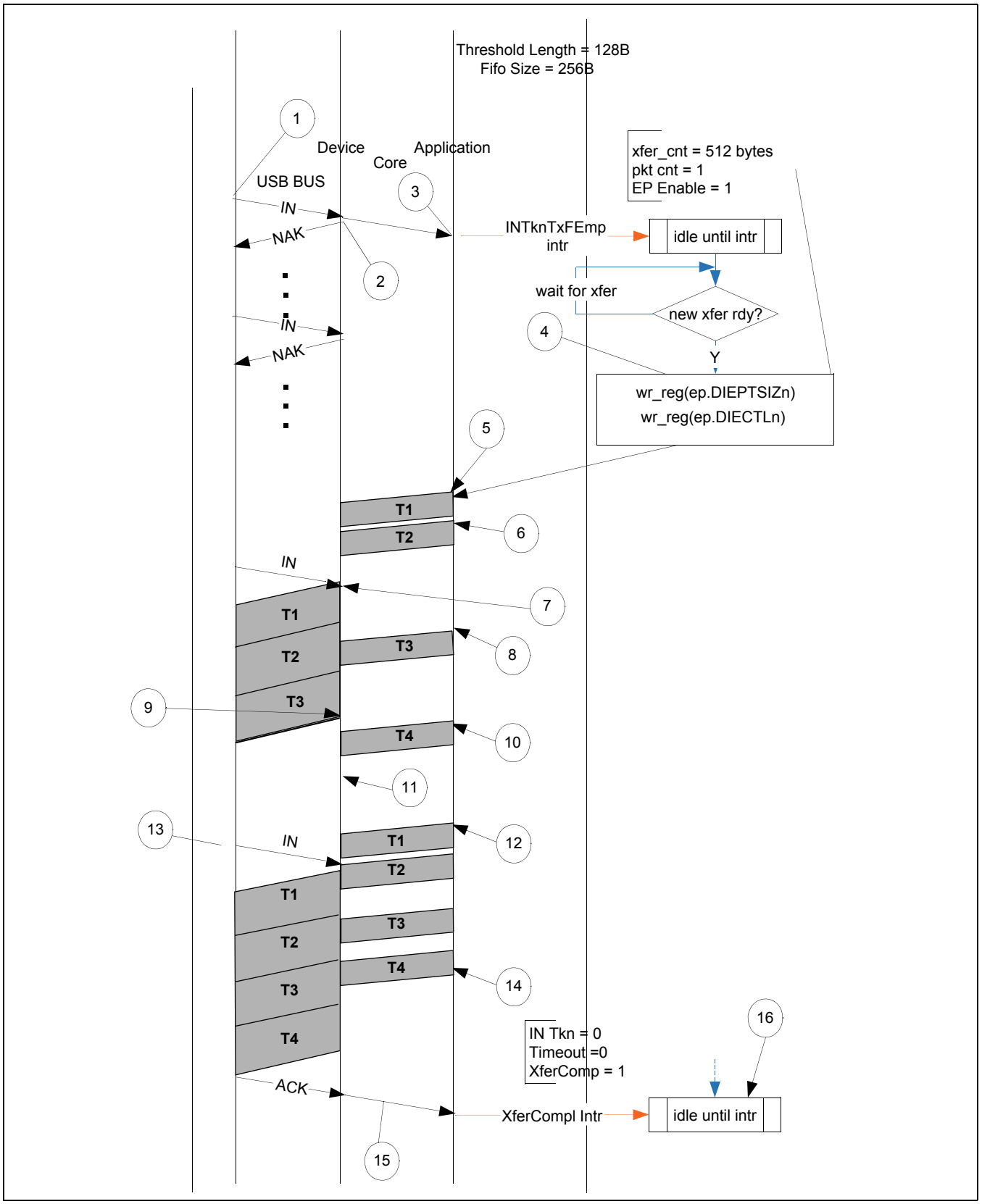


Bulk IN DMA mode With Thresholding

These notes refer to *Figure 38-95* on page 1482.

1. The Host attempts to read data (IN token) from an Endpoint
2. On receiving the IN token on the USB bus, the function sends back a NAK handshake because no data is available in the Transmit FIFO
3. To indicate to the application that there was no data to send, the function generates a DIEPINTn.IN Token Rcvd When TxF Empty interrupt.
4. When data is ready, the application sets up the DIEPTSIZn register with the TransferSize and Packet Count fields and enables the Endpoint.
5. On seeing the Endpoint enabled, the internal DMA engine start fetching the first threshold amount of data.
6. DMA engine fetching the second threshold.
7. The Host re-attempts the IN token, and the function start sending the data because it sees at least threshold amount of data in the FIFO.
8. The DMA engine starts fetching the third threshold after it sees threshold amount of space.
9. The MAC sees an under run condition because of FIFO being empty in the middle of the packet, stops the packet and corrupt the CRC.
10. The DMA engine starts to fetch the last threshold for that packet but it is late and will eventually result in under run.
11. No handshake response from the Host, because the packet was corrupted. The function rewinds the pointers and flush the FIFO.
12. The function starts to re-fetch the packet, staring with the first threshold.
13. The Host re-attempts the IN token, and the function starts sending the data, since the function detects at least the threshold amount of data in the FIFO.
14. The function fetches the last threshold in time.
15. The function receives the handshake from Host and Because the XferSize is now zero, the intended transfer is complete. The Device function generates a DIEPINTn.XferCompl interrupt.
16. The application processes the interrupt and uses the setting of DIEPINTn.XferCompl interrupt bit to determine that the intended transfer is complete.

Figure 38-95. Bulk IN DMA mode with Thresholding

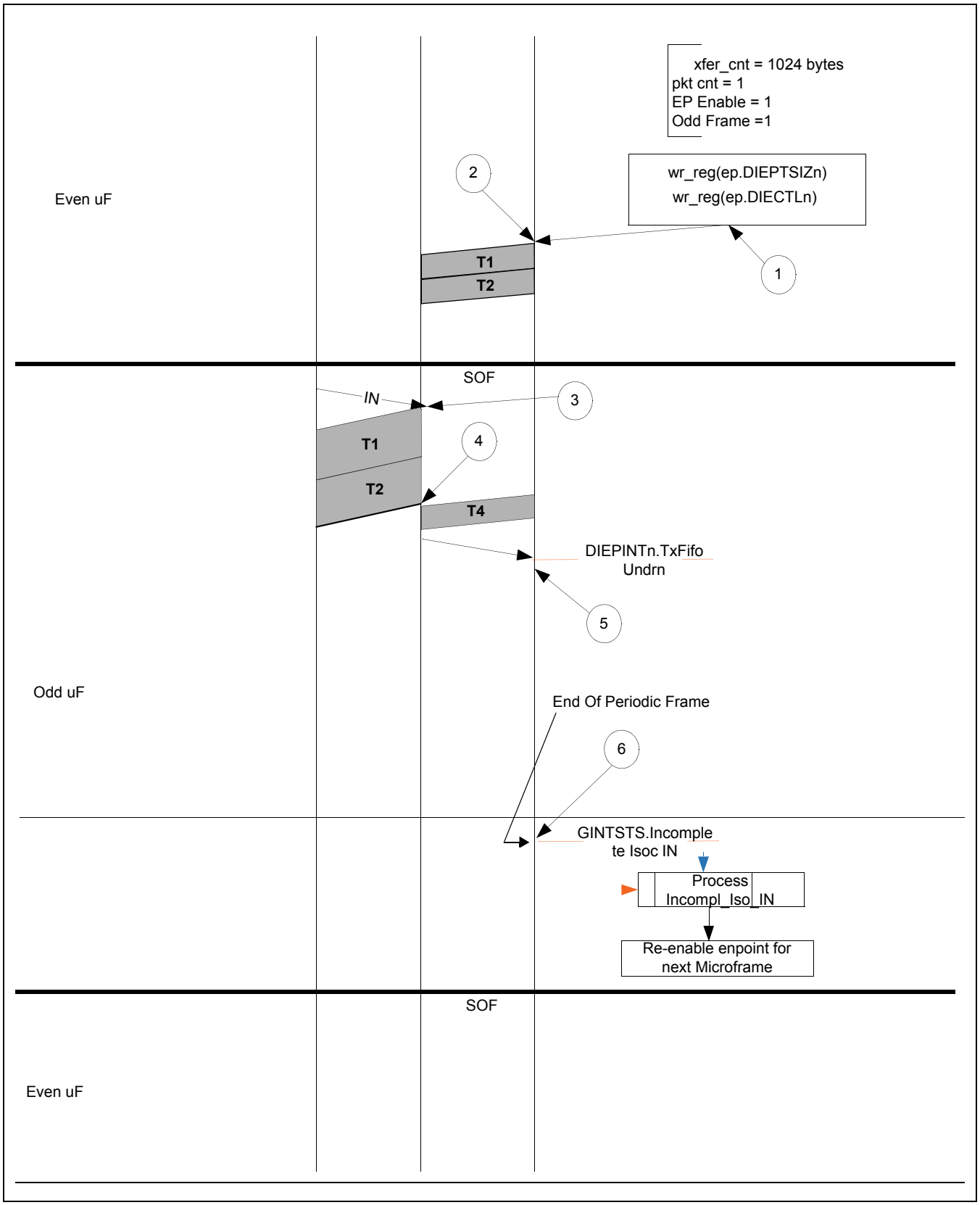


Isochronous IN DMA Mode With Thresholding

Figure 38-96 on page 1484 shows an isochronous In transfer when thresholding is enabled. The FIFO size is assumed to be 512B and the threshold length is 128B.

1. The application enabled the isochronous IN Endpoint for Odd microframe.
2. The function starts fetching the first two thresholds.
3. The function starts sending data out in response to the IN token.
4. The function sees an under run condition, because the third threshold was not fetched in time.
5. The function generates DIEPINTn.TXFifoUnderrun interrupt (In this case, the application ignores this interrupt).
6. At the End of Periodic Frame Interval, the function generates GINTSTS.IncomplISO interrupt.

Figure 38-96. Isochronous IN DMA Mode with Thresholding



38.10.2.3 Control Transfers

This section describes the various types of control transfers.

Control Write Transfers (SETUP, Data OUT, Status IN)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTn.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See *SETUP Transactions* on page 1448 for more details. At the end of the Setup stage, the application must reprogram the DOEPTSIZn.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data OUT phase, program the function to perform a control OUT transfer as explained in *Generic Non-Isochronous OUT Data Transfers Without Thresholding* on page 1452.

   In DMA mode, the application must reprogram the DOEPDMAn register to receive a control OUT data packet to a different memory location.
3. In a single OUT data transfer on control Endpoint 0, the application can receive up to 64B. If the application is expecting more than 64B in the Data OUT stage, the application must re-enable the Endpoint to receive another 64B, and must continue to do so until it has received all the data in the Data stage.
4. Assertion of the DOEPINTn.Transfer Compl interrupt on the last data OUT transfer indicates the completion of the data OUT phase of the control transfer.
5. On completion of the data OUT phase, the application must do the following.
   - To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT Endpoint as explained in section in section *SETUP Transactions* on page 1448.
     – DOEPCTLn.EPEna = 1'b1
   - To execute the received Setup command, the application must program the required registers in the function. This step is optional, based on the type of Setup command received.
6. For the status IN phase, the application must program the function as described in *Generic Non-periodic IN Data Transfers Without Thresholding* on page 1464 to perform a data IN transfer.
   - At this point, when Shared FIFO operation is used, and IN endpoints other than the control Endpoint are enabled to perform data transfers, or when there is data belonging to other IN endpoints already posted into the Non-periodic Transmit FIFO, the application must first disable these other endpoints and flush the Non-periodic Transmit FIFO before posting the control Endpoint's status IN data to the Non-periodic Transmit FIFO.
7. Assertion of the DIEPINTn.Transfer Compl interrupt indicates completion of the status IN phase of the control transfer.
8. If the DIEPINTn.TIMEOUT interrupt is detected before transfer completion, the application must follow the steps in *Timeout for Non-periodic IN Data Transfers: Shared FIFO* on page 1472.
9. The previous step must be repeated until the DIEPINTn.Transfer Compl interrupt is detected on the Endpoint, marking the completion of the control write transfer.

Control Read Transfers (SETUP, Data IN, Status OUT)

This section describes control write transfers.

Application Programming Sequence

1. Assertion of the DOEPINTn.SETUP Packet interrupt indicates that a valid SETUP packet has been transferred to the application. See *SETUP Transactions* on page 1448 for more details. At the end of the Setup stage, the application must reprogram the DOEPTSIZn.SUPCnt field to 3 to receive the next SETUP packet.
2. If the last SETUP packet received before the assertion of the SETUP interrupt indicates a data IN phase, program the function to perform a control IN transfer as explained in *Generic Non-periodic IN Data Transfers Without Thresholding* on page 1464.
3. On a single IN data transfer on control Endpoint 0, the application can transmit up to 64B. To transmit more than 64B in the Data IN stage, the application must re-enable the Endpoint to transmit another 64B, and must continue to do so, until it has transmitted all the data in the Data stage.
4. If the DIEPINTn.TIMEOUT interrupt is detected before transfer completion, the application must follow the steps in *Timeout for Non-periodic IN Data Transfers: Shared FIFO* on page 1472.
5. The previous step must be repeated until the DIEPINTn.Transfer Compl interrupt is detected for every IN transfer on the Endpoint.
6. The DIEPINTn.Transfer Compl interrupt on the last IN data transfer marks the completion of the control transfer's Data stage.
7. To perform a data OUT transfer in the status OUT phase, the application must program the function as described in *SETUP and OUT Data Transfers* on page 1446.
   - The application must program the DCFG.NZStsOUTHShk handshake field to a proper setting before transmitting an data OUT transfer for the Status stage.
   - In DMA mode, the application must reprogram the DOEPDMAn register to receive the control OUT data packet to a different memory location.
8. Assertion of the DOEPINTn.Transfer Compl interrupt indicates completion of the status OUT phase of the control transfer. This marks the successful completion of the control read transfer.
   - To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT Endpoint as explained in *SETUP Transactions* on page 1448.
     – DOEPCTLn.EPEna = 1'b1

Two-Stage Control Transfers (SETUP/Status IN)

This section describes two-stage control transfers.

Application Programming Sequence

1. Assertion of the DOEPINTn.SetUp interrupt indicates that a valid SETUP packet has been transferred to the application. See *SETUP Transactions* on page 1448 for more detail. To receive the next SETUP packet, the application must reprogram the DOEPTSIZn.SUPCnt field to 3 at the end of the Setup stage.
2. Decode the last SETUP packet received before the assertion of the SETUP interrupt. If the packet indicates a two-stage control command, the application must do the following.

   To transfer a new SETUP packet in DMA mode, the application must re-enable the control OUT Endpoint. See *SETUP Transactions* on page 1448 for details.
   - DOEPCTLn.EPEna = 1'b1
   - Depending on the type of Setup command received, the application can be required to program registers in the function to execute the received Setup command.

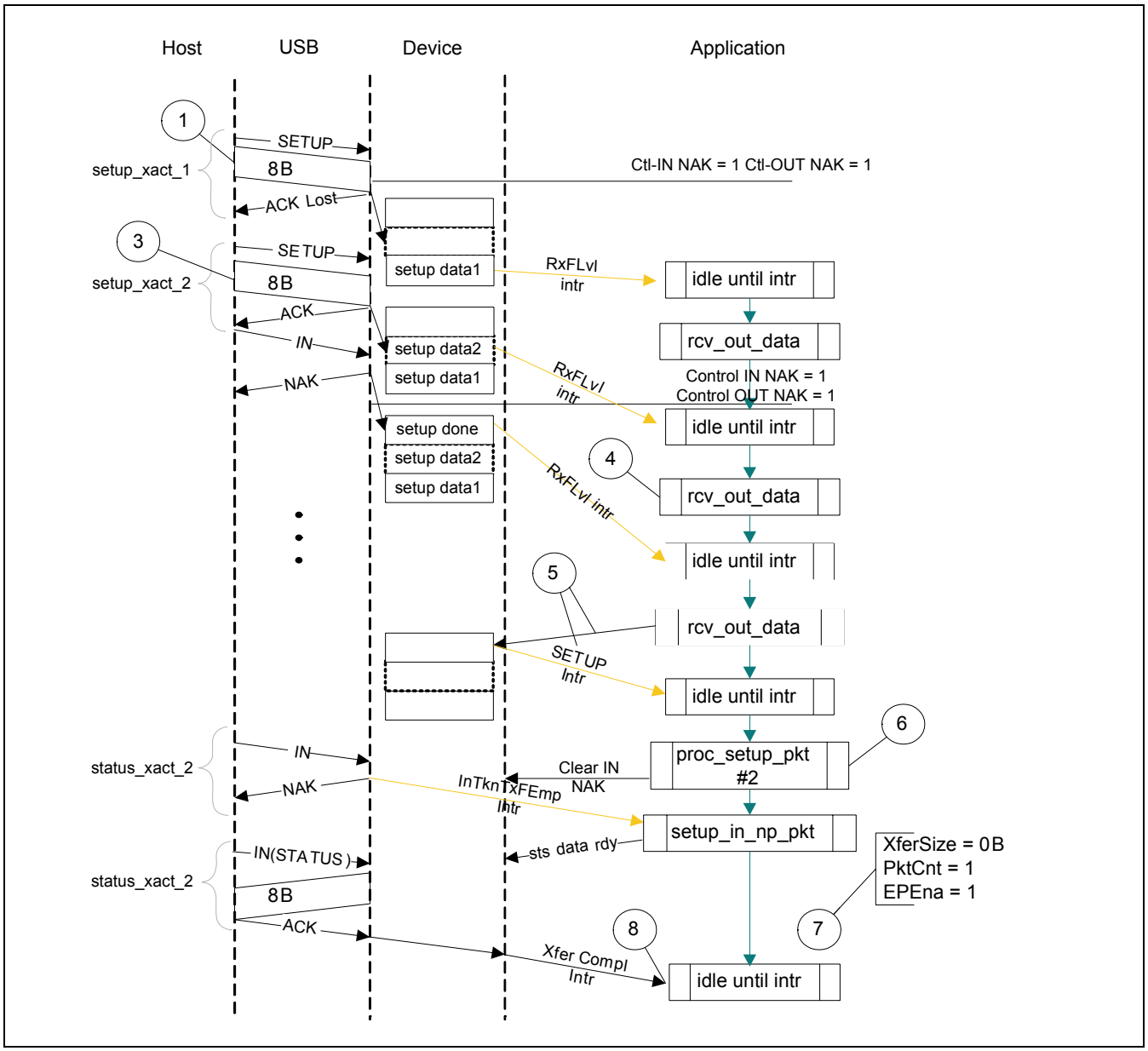
3. For the status IN phase, the application must program the function described in *Generic Non-periodic IN Data Transfers Without Thresholding* on page 1464 to perform a data IN transfer.
4. Assertion of the DIEPINTn.Transfer Compl interrupt indicates the completion of the status IN phase of the control transfer.
5. If the DIEPINTn.TIMEOUT interrupt is detected before the transfer completion, the application must follow the steps in *Timeout for Non-periodic IN Data Transfers: Shared FIFO* on page 1472.
6. The previous step must be repeated until the DIEPINTn.Transfer Compl interrupt is detected on the Endpoint, marking the completion of the two-stage control transfer.

Example: Two-Stage Control Transfer

These notes refer to *Figure 38-97* on page 1488.

1. SETUP packet #1 is received on the USB and is written to the receive FIFO, and the function responds with an ACK handshake. This handshake is lost and the Host detects a timeout.
2. The SETUP packet in the receive FIFO results in a GINTSTS.RxFLvl interrupt to the application, causing the application to empty the receive FIFO.
3. SETUP packet #2 on the USB is written to the receive FIFO, and the function responds with an ACK handshake.
4. The SETUP packet in the receive FIFO sends the application the GINTSTS.RxFLvl interrupt and the application empties the receive FIFO.
5. After the second SETUP packet, the Host sends a control IN token for the status phase. The function issues a NAK response to this token, and writes a Setup Stage Done entry to the receive FIFO. This entry results in a GINTSTS.RxFLvl interrupt to the application, which empties the receive FIFO. After reading out the Setup Stage Done DWORD, the function asserts the DOEPINTn.SetUp packet interrupt to the application.
6. On this interrupt, the application processes SETUP Packet #2, decodes it to be a two-stage control command, and clears the control IN NAK bit.
   - DIEPCTLn.CNAK = 1
7. When the application clears the IN NAK bit, the function interrupts the application with a DIEPINTn.INTknTXFEmp. On this interrupt, the application enables the control IN Endpoint with a DIEPTSIZn.XferSize of 0 and a DIEPTSIZn.PktCnt of 1. This results in a zero-length data packet for the status IN token on the USB.
8. At the end of the status IN phase, the function interrupts the application with a DIEPINTn.XferCompl interrupt.

Figure 38-97. Two-Stage Control Transfer



38.10.2.4 Endpoint Mismatch Handling

This section is valid only for Shared FIFO operations.

Endpoint Mismatch

When shared FIFO operation is chosen, the function has a single non periodic transmit FIFO to store IN data for all non-periodic endpoints. Consequently, the application must predict the sequence in which the

USB Host accesses the non-periodic IN endpoints in the Device and posts the data for these endpoints in the Non-periodic Transmit FIFO accordingly.

If the application fails to predict this sequence and posts the data for a different Endpoint than the one that the USB Host accesses, an Endpoint mismatch in the Non-periodic Transmit FIFO results, and the function responds to these IN tokens with a NAK handshake and interrupts the application with the following two interrupts:

- DIEPINTn.INTknEPMis, when an IN token is received and the Endpoint number in the top entry of the Non-periodic Transmit FIFO does not match the Endpoint number for which the IN token is received.
- GINTSTS.EPMis, when the IN Token Sequence Learning Queue exceeds the application-programmed EPMismatch counter value.

Learning Queue Implementation and Usage

If the application cannot predict the USB Host's non-periodic IN Endpoint access sequence through a higher-level protocol, it can use the IN Token Sequence Learning Queue to observe this access sequence. Key features of this learning queue are as follows.

- The function writes the Endpoint number to this learning queue every time the USB Host accesses a bulk, control or non-periodic interrupt Endpoint, irrespective of the handshake sent by the function.
- The function continues writing to the learning queue when it is full, overwriting old contents of the queue.
- The function implements a counter whose maximum value is programmed to the field DCFG.EPMisCnt. This counter is decremented on every IN token Endpoint mismatch. The counter is restored to the application-programmed maximum value whenever there is an Endpoint match or when the counter reaches 0.
- When this counter reaches 0, indicating that application-programmed consecutive Endpoint mismatches have occurred, the function asserts the GINTSTS.EPMis interrupt.
- The IN token sequence can be determined by leaving the Non-periodic Transmit FIFO empty and examining the IN token sequence in the learning queue. The application can read, pop and flush this learning queue using the following registers and register fields.
  - DTKNQR1: Device IN Token Queue Read register 1
  - DTKNQR2: Device IN Token Queue Read register 2
  - DTKNQP: Device IN Token Queue Pop register
  - DRSTCTL. IN Token Sequence Learning Queue Flush
  - DINTSTS.IN Token Received Interrupt

Non-periodic IN Endpoint Sequencing

In DMA mode, the DIEPCTLn.NextEp value programmed controls the order in which the function fetches non-periodic data for IN endpoints.

If application requires the function to fetch data for the non-periodic IN endpoints in a certain Endpoint order, it must program the DIEPCTLn.NextEP field accordingly before enabling the endpoints. To enable a single Endpoint enabled at a time the application must set the DIEPCTLn.NextEP field to the Endpoint number itself. The function uses the NextEP field irrespective of the DIEPCTLn.EPEna bit.

38.10.3 Handling Babble Conditions

If the USB receives a packet that is larger than the maximum packet size for that Endpoint, the function stops writing data to the Rx buffer and waits for the end of packet (EOP). When the function detects the EOP, it flushes the packet in the Rx buffer and does not send any response to the Host.

If the function continues to receive data at the EOF2 (the end of frame 2, which is very close to SOF), the function generates an early_suspend interrupt (GINTSTS.ErlySusp). On receiving this interrupt, the application must check the erratic_error status bit (DSTS.ErrticErr). If this bit is set, the application must take it as a long babble and perform a soft reset.

38.10.4 Worst Case Response Time

When the USB acts as a Device, there is a worst case response time for any tokens that follow an isochronous OUT. This worst case response time depends on the AHB clock frequency.

The function registers are in the AHB domain, and the function does not accept another token before updating these register values. The worst case is for any token following an isochronous OUT, because for an isochronous transaction, there is no handshake and the next token could come sooner. This worst case value is 7 PHY clocks when the AHB clock is the same as the PHY clock. When AHB clock is faster, this value is smaller.

If this worst case condition occurs, the function responds to bulk/interrupt tokens with a NAK and drops isochronous and SETUP tokens. The Host interprets this as a timeout condition for SETUP and retries the SETUP packet. For isochronous transfers, the incomplISOCIN and incomplISOCOUT interrupts inform the application that isochronous IN/OUT packets were dropped.

38.10.5 Choosing the Value of GUSBCFG.USBTrdTim

The value in GUSBCFG.USBTrdTim is the time it takes for the MAC, in terms of PHY clocks after it has received an IN token, to get the FIFO status, and thus the first data from PFC (Packet FIFO Controller) block. This time involves the synchronization delay between the PHY and AHB clocks. The worst case delay for this is when the AHB clock is the same as the PHY clock. In this case, the delay is 5 clocks.

Once the MAC receives an IN token, this information (token received) is synchronized to the AHB clock by the PFC (the PFC runs on the AHB clock). The PFC then reads the data from the SPRAM and writes it into the dual clock source buffer. The MAC then reads the data out of the source buffer (four deep).

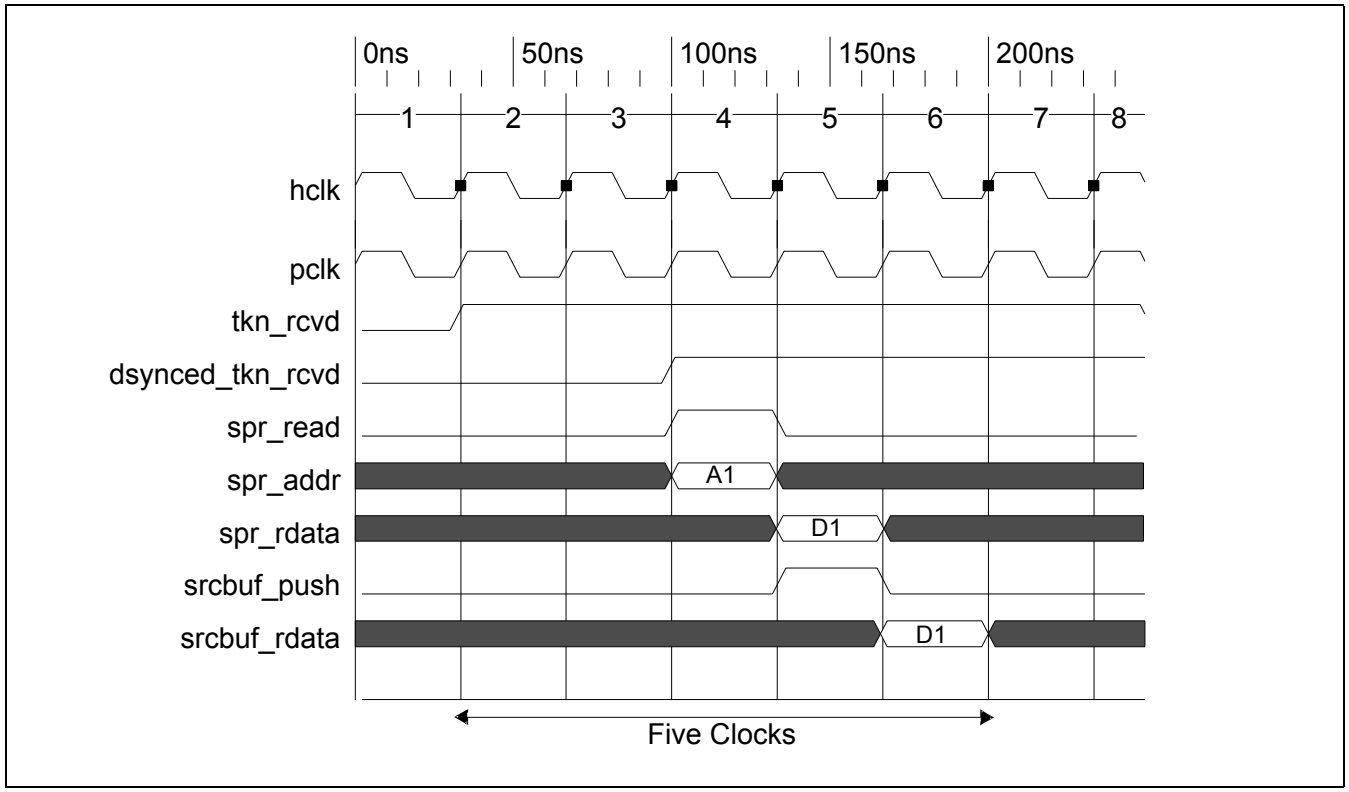
If the AHB is running at a higher frequency than the PHY, the application can use a smaller value for GUSBCFG.USBTrdTim. *Figure 38-98* on page 1491 explains the 5-clock delay. This diagram has the following signals:

- tkn_rcvd: Token received information from MAC to PFC
- dynced_tkn_rcvd: Doubled sync tkn_rcvd, from pclk to hclk domain
- spr_read: Read to SPRAM
- spr_addr: Address to SPRAM
- spr_rdata: Read data from SPRAM
- srcbuf_push: Push to the source buffer
- srcbuf_rdata: Read data from the source buffer. Data seen by MAC

The application can use the following formula to calculate the value of GUSBCFG.USBTrdTim:

4 * AHB Clock + 1 PHY Clock
= (2 clock sync + 1 clock memory address + 1 clock memory data from sync RAM) +
(1 PHY Clock (next PHY clock MAC can sample the 2-clock FIFO output)

Figure 38-98. USBTrdTim Max Timing Case

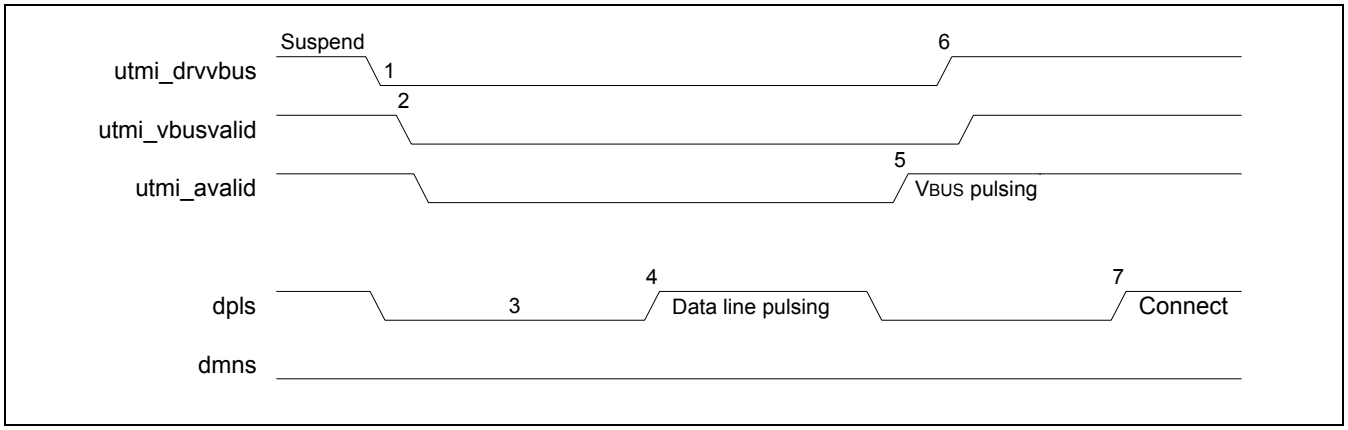


38.11 OTG Programming Model

The USB function is an OTG Device supporting HNP and SRP. When the function is connected to an "A" plug, it is referred to as an A-Device. When the function is connected to a "B" plug it is referred to as a B-Device. In Host mode, the USB function turns off V_{BUS} to conserve power. SRP is a method by which the B-Device signals the A-Device to turn on V_{BUS} power. A Device must perform both data-line pulsing and V_{BUS} pulsing, but a Host can detect either data-line pulsing or V_{BUS} pulsing for SRP. HNP is a method by which the B-Device negotiates and switches to Host role. In Negotiated mode after HNP, the B-Device suspends the bus and reverts to the Device role.

38.11.1 A-Device Session Request Protocol (SRP)

Figure 38-99. A-Device SRP

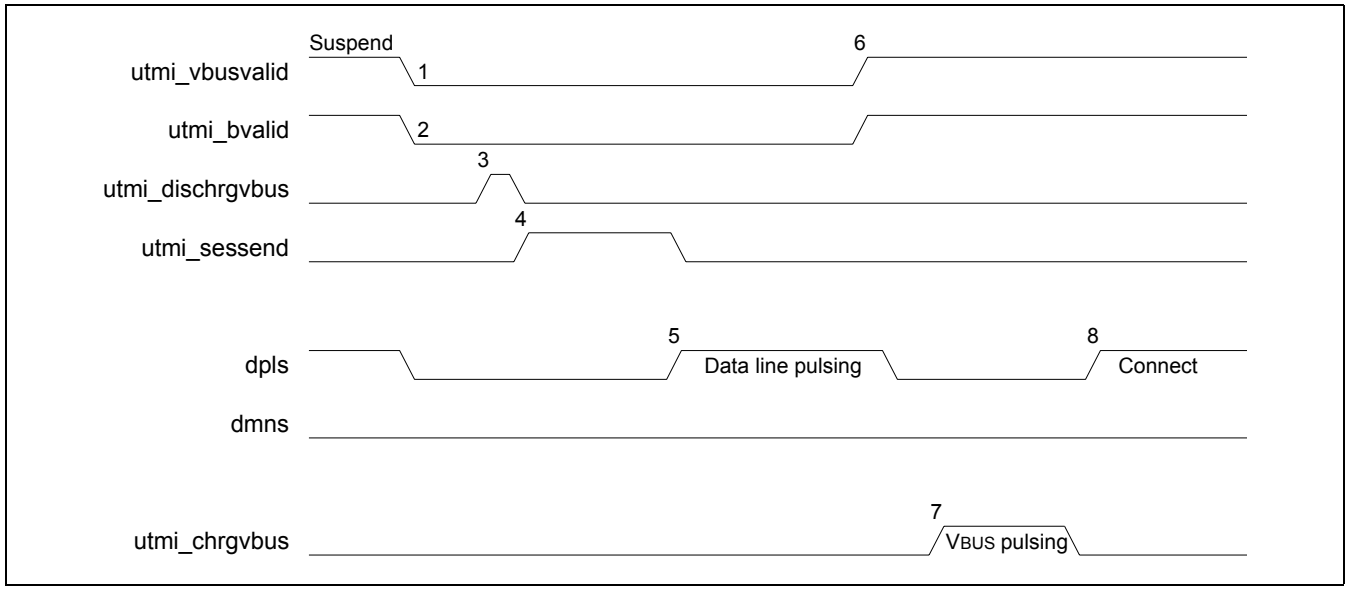


The application must set the SRP-Capable bit in the function USB Configuration register. This enables the USB function to detect SRP as an A-Device.

1. To save power, the application suspends and turns off port power when the bus is idle by writing the Port Suspend and Port Power bits in the Host Port Control and Status register.
2. PHY indicates port power off by deasserting the utmi_vbusvalid signal.
3. The Device must detect SE0 for at least 2 ms to start SRP when V_{BUS} power is off.
4. To initiate SRP, the Device turns on its data line pull-up resistor for 5 to 10 ms. The USB function detects data-line pulsing.
5. The Device drives V_{BUS} above the A-Device session valid (2.0 V minimum) for V_{BUS} pulsing.
   The USB function interrupts the application on detecting SRP. The Session Request Detected bit is set in the OTG Interrupt Status register.
6. The application must service the Session Request Detected interrupt and turn on the Port Power bit by writing the Port Power bit in the Host Port Control and Status register. The PHY indicates port power-on by asserting utmi_vbusvalid signal.
7. When the USB is powered, the Device connects, completing the SRP process.

38.11.2 B-Device Session Request Protocol (SRP)

Figure 38-100. B-Device SRP

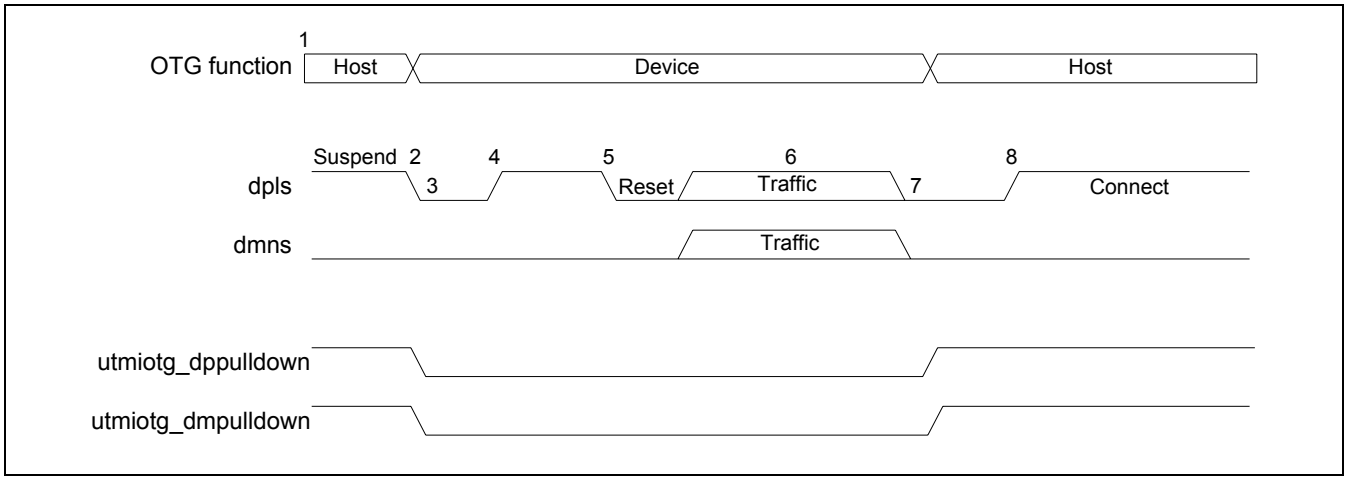


The application must set the SRP-Capable bit in the function USB Configuration register. This enables the USB function to initiate SRP as a B-Device. SRP is a means by which the USB function can request a new session from the Host.

1. To save power, the Host suspends and turns off port power when the bus is idle. PHY indicates port power off by deasserting the utmi_vbusvalid signal.
   The USB function sets the Early Suspend bit in the function Interrupt register after 3 ms of bus idleness. Following this, the USB function sets the USB Suspend bit in the function Interrupt register.
2. The PHY indicates the end of the B-Device session by deasserting the utmi_bvalid signal.
3. The USB function asserts the utmi_dischrgvbus signal to indicate to the PHY to speed up V_{BUS} discharge.
4. The PHY indicates the session's end by asserting the utmi_sessend signal. This is the initial condition for SRP. The USB function requires 2 ms of SE0 before initiating SRP.
   For a USB 1.1 full-speed serial transceiver, the application must wait until V_{BUS} discharges to 0.2 V after GOTGCTL.BSesVld is deasserted. This discharge time can be obtained from the transceiver vendor and varies between different transceivers.
5. The application initiates SRP by writing the Session Request bit in the OTG Control and Status register. The USB function perform data-line pulsing followed by V_{BUS} pulsing.
6. The Host detects SRP from either the data-line or V_{BUS} pulsing, and turns on V_{BUS}. The PHY indicates V_{BUS} power-on by asserting utmi_vbusvalid.
7. The USB function performs V_{BUS} pulsing by asserting utmi_chrgvbus.
   The Host starts a new session by turning on V_{BUS}, indicating SRP success. The USB function interrupts the application by setting the Session Request Success Status Change bit in the OTG Interrupt Status register. The application reads the Session Request Success bit in the OTG Control and Status register.
8. When the USB is powered, the USB function connects, completing the SRP process.

38.11.3 A-Device Host Negotiation Protocol

Figure 38-101. A-Device HNP

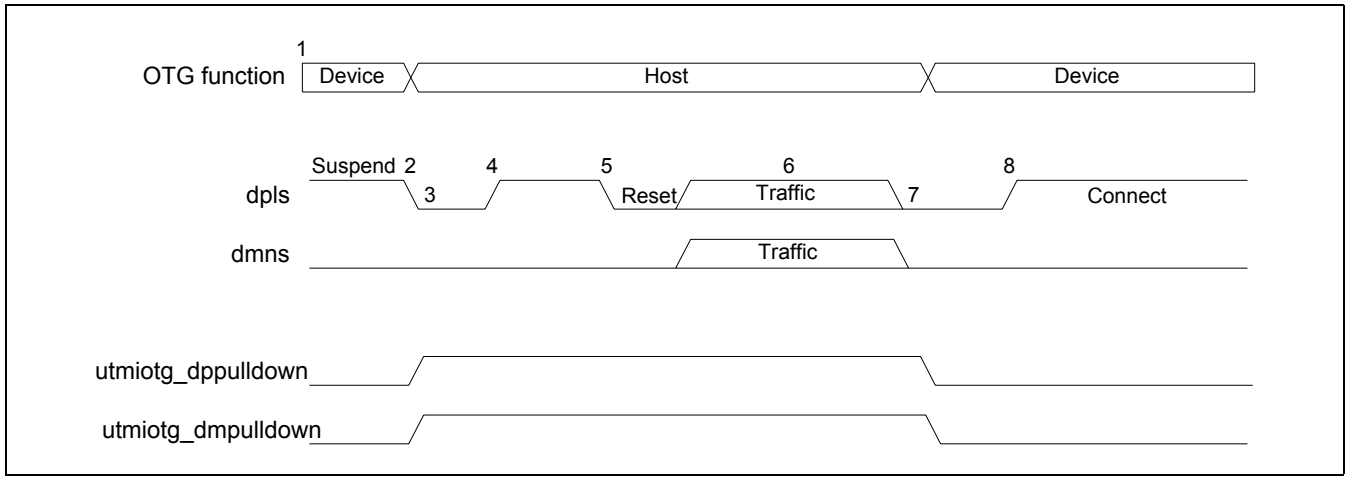


HNP switches the USB Host role from the A-Device to the B-Device. The application must set the HNP-Capable bit in the function USB Configuration register to enable the USB function to perform HNP as an A-Device.

1. The USB function sends the B-Device a SetFeature b_hnp_enable descriptor to enable HNP support. The B-Device's ACK response indicates that the B-Device supports HNP. The application must set Host Set HNP Enable bit in the OTG Control and Status register to indicate to the USB function that the B-Device supports HNP.
2. When it has finished using the bus, the application suspends by writing the Port Suspend bit in the Host Port Control and Status register.
3. When the B-Device observes a USB suspend, it disconnects, indicating the initial condition for HNP. The B-Device initiates HNP only when it must switch to the Host role; otherwise, the bus continues to be suspended. The USB function sets the Host Negotiation Detected interrupt in the OTG Interrupt Status register, indicating the start of HNP.
4. The USB function deasserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate a Device role. The PHY enable the D + pull-up resistor indicates a connect for B-Device.
   The application must read the Current Mode bit in the OTG Control and Status register to determine Device mode operation.
5. The B-Device detects the connection, issues a USB reset, and enumerates the USB function for data traffic.
6. The B-Device continues the Host role, initiating traffic, and suspends the bus when done.
   The USB function sets the Early Suspend bit in the function Interrupt register after 3 ms of bus idleness. Following this, the USB function sets the USB Suspend bit in the function Interrupt register.
7. In Negotiated mode, the USB function detects the suspend, disconnects, and switches back to the Host role. The USB function asserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate its assumption of the Host role.
   The USB function sets the Connector ID Status Change interrupt in the OTG Interrupt Status register. The application must read the connector ID status in the OTG Control and Status register to determine the USB function's operation as an A-Device. This indicates the completion of HNP to the application. The application must read the Current Mode bit in the OTG Control and Status register to determine Host mode operation.
8. The B-Device connects, completing the HNP process.

38.11.4 B-Device Host Negotiation Protocol

Figure 38-102. B-Device HNP



HNP switches the USB Host role from B-Device to A-Device. The application must set the HNP-Capable bit in the function USB Configuration register to enable the USB function to perform HNP as a B-Device.

1. The A-Device sends the SetFeature b_hnp_enable descriptor to enable HNP support. The USB function's ACK response indicates that it supports HNP. The application must set the Device HNP Enable bit in the OTG Control and Status register to indicate HNP support.
   The application sets the HNP Request bit in the OTG Control and Status register to indicate to the USB function to initiate HNP.
2. When it has finished using the bus, the A-Device suspends by writing the Port Suspend bit in the Host Port Control and Status register. The USB function sets the Early Suspend bit in the function Interrupt register after 3 ms of bus idleness. Following this, the USB function sets the USB Suspend bit in the function Interrupt register.
3. The USB function disconnects and the A-Device detects SE0 on the bus, indicating HNP. The USB function asserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate its assumption of the Host role.
4. The A-Device responds by activating its D + pull-up resistor within 3 ms of detecting SE0. The USB function detects this as a connect.
   The USB function sets the Host Negotiation Success Status Change interrupt in the OTG Interrupt Status register, indicating the HNP status. The application must read the Host Negotiation Success bit in the OTG Control and Status register to determine Host negotiation success. The application must read the Current Mode bit in the function Interrupt register (GINTSTS) to determine Host mode operation.
5. The USB function issues a USB reset and enumerates the A-Device for data traffic.
6. The USB function continues the Host role of initiating traffic, and when done, suspends the bus by writing the Port Suspend bit in the Host Port Control and Status register.
7. In Negotiated mode, when the A-Device detects a suspend, it disconnects and switches back to the Host role. The USB function deasserts the utmiotg_dppulldown and utmiotg_dmpulldown signals to indicate the assumption of the Device role.
   The application must read the Current Mode bit in the function Interrupt (GINTSTS) register to determine the Host mode operation.
8. The USB function connects, completing the HNP process.

38.12 Clock Gating Programming Model

When the USB is suspended or the session is not valid, the PHY is driven into Suspend mode, stopping the PHY clock to reduce power consumption in the PHY and the OTG function. To further reduce power consumption, the OTG function also supports AHB clock gating. You can gate the AHB clock to some of the OTG modules when the USB is suspended or the session in not valid.

This section describes the programming sequence for using those power-saving features.

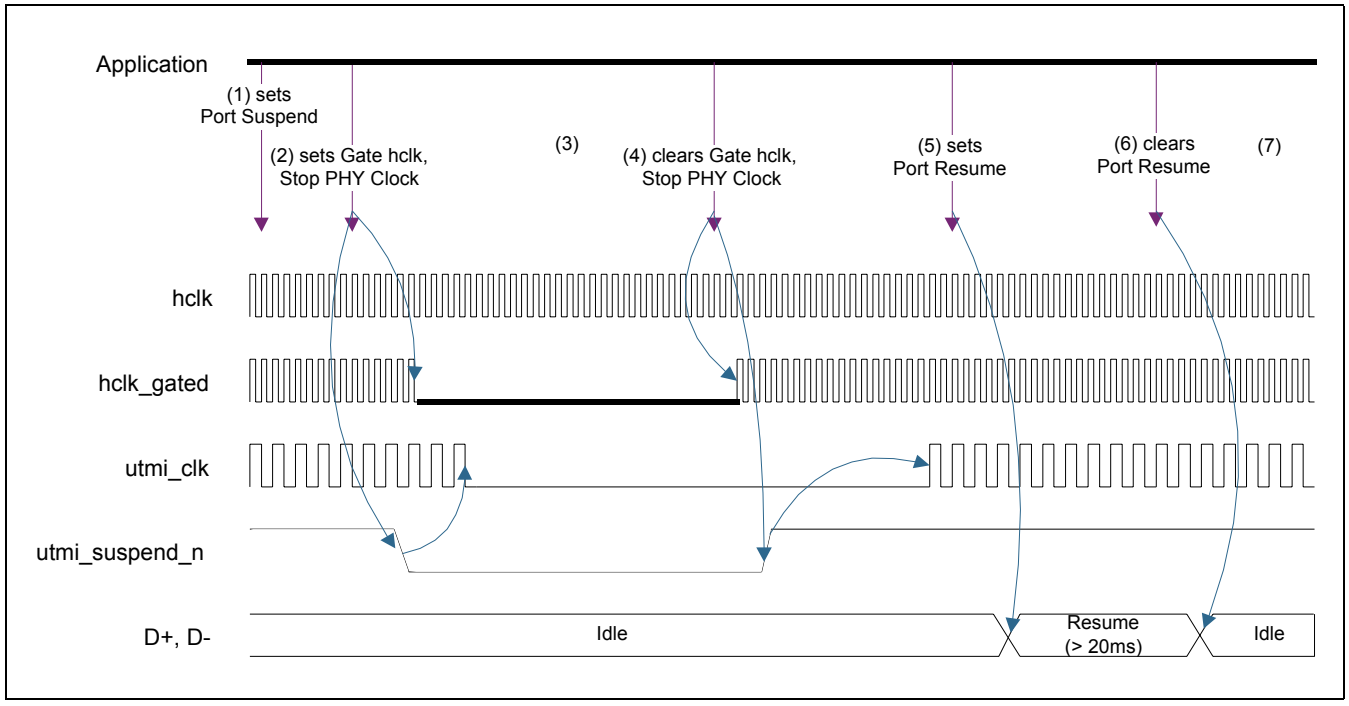
38.12.1 Clock Gating

You can use clock gating to reduce power consumption when the USB is suspended or the session is not valid. The PHY turns off the PHY clock for as long as the function asserts the suspend_n signal to the PHY. The AHB clock to the OTG internal modules can also be gated by writing to the Gate hclk bit in the Power and Clock Gating Control register. The following sections show the procedures you must follow to use the clock gating feature.

38.12.1.1 Host Mode Suspend and Resume With Clock Gating

1. The application sets the Port Suspend bit in the Host Port CSR, and the function drives a USB suspend.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, the function gates the hclk (hclk_gated) to AHB-domain modules other than the BIU.
3. The function remains in Suspend mode.
4. The application clears the Gate hclk and Stop PHY Clock bits, and the PHY clock is generated.
5. The application sets the Port Resume bit, and the function starts driving Resume signaling.
6. The application clears the Port Resume bit after at least 20 ms.
7. The function is in normal operating mode.

Figure 38-103. Host Mode Suspend and Resume With Clock Gating

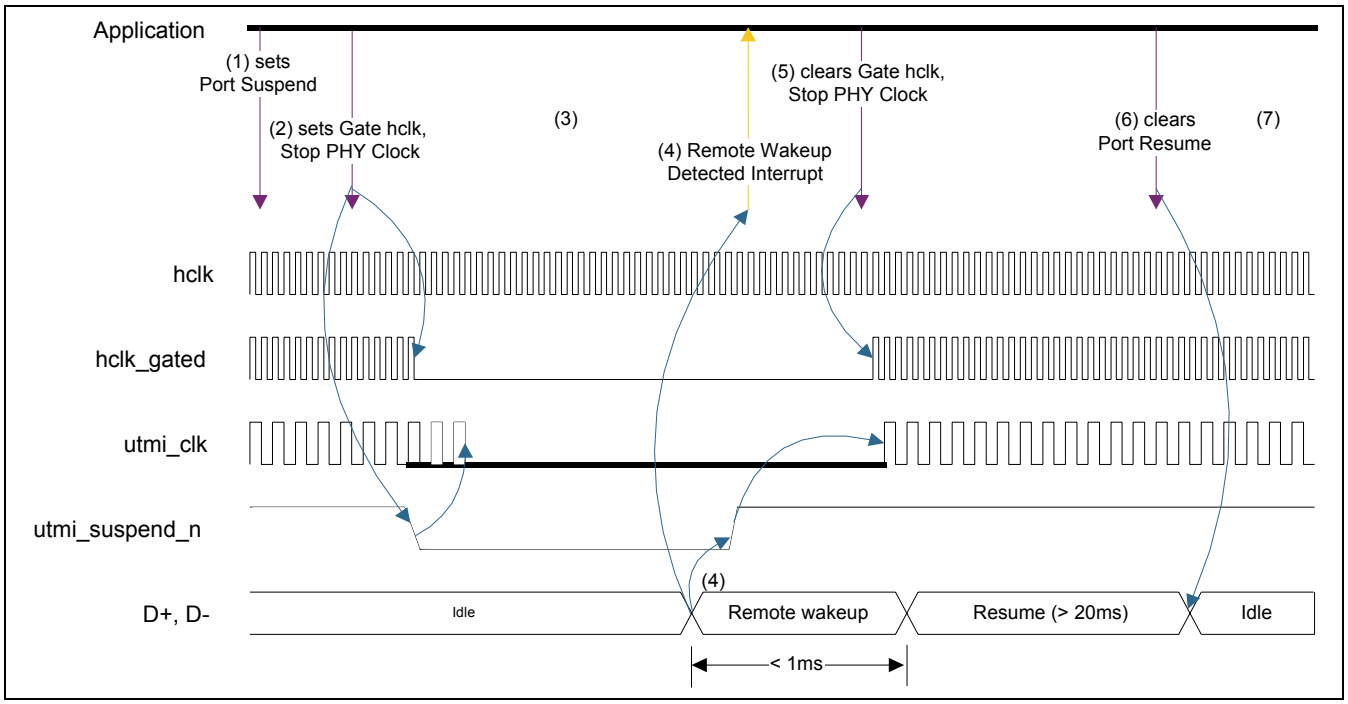


38.12.1.2 Host Mode Suspend and Remote Wake Up With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the function drives a USB suspend.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the function gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
3. The function remains in Suspend mode.
4. The Remote Wake Up signaling from the Device is detected. The function deasserts the suspend_n signal to the PHY to generate the PHY clock. The function generates a Remote Wake Up Detected interrupt.
5. The application clears the Gate hclk and Stop PHY Clock bits. The function sets the Port Resume bit.
6. The application clears the Port Resume bit after at least 20 ms.
7. The function is in normal operating mode.

Figure 38-104. Host Mode Suspend and Remote Wakeup With Clock Gating



38.12.1.3 Host Mode Session End and Start With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the function drives a USB suspend.
2. The application clears the Port Power bit. The function turns off V_{BUS}.
3. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the function gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
4. The function remains in Low-Power mode.
5. The application clears the Gate hclk bit and the application clears the Stop PHY Clock bit to start the PHY clock.
6. The application sets the Port Power bit to turn on V_{BUS}.
7. The function detects Device connection and drives a USB reset.
8. The function is in normal operating mode.

38.12.1.4 Host Mode Session End and SRP With Clock Gating

Sequence of operations:

1. The application sets the Port Suspend bit in the Host Port CSR, and the function drives a USB suspend.
2. The application clears the Port Power bit. The function turns off V_{BUS}.
3. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the

Power and Clock Gating Control register, and the function gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.

4. The function remains in Low-Power mode.
5. SRP (data line pulsing) from the Device is detected. The function deasserts the suspend_n signal to the PHY to generate the PHY clock. An SRP Request Detected interrupt is generated.
6. The application clears the Gate hclk bit and the Stop PHY Clock bit.
7. The function sets the Port Power bit to turn on V_{BUS}.
8. The function detects Device connection and drives a USB reset.
9. The function is in normal operating mode.

38.12.1.5 Device Mode Suspend and Resume With Clock Gating

Sequence of operations:

1. The function detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the function gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
3. The function remains in Suspend mode.
4. The Resume signaling from the Host is detected. The function deasserts the suspend_n signal to the PHY to generate the PHY clock. A Resume Detected interrupt is generated.
5. The application clears the Gate hclk bit and the Stop PHY Clock bit.
6. The Host finishes Resume signaling.
7. The function is in normal operating mode.

38.12.1.6 Device Mode Suspend and Remote Wake Up With Clock Gating

Sequence of operations:

1. The function detects a USB suspend and generates a Suspend Detected interrupt.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, the function gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
3. The function remains in Suspend mode.
4. The application clears the Gate hclk bit and the Stop PHY Clock bit.
5. The application sets the Remote Wakeup bit in the Device Control register, the function starts driving Remote Wake Up signaling.
6. The Host drives Resume signaling.
7. The function is in normal operating mode.

38.12.1.7 Device Mode Session End and Start With Clock Gating

Sequence of operations:

1. The function detects a USB suspend, and generates a Suspend Detected interrupt. The Host turns off V_{BUS}.

2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the function gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
3. The function remains in Low-Power mode.
4. The new session is detected (bsessvld is high). The function deasserts the suspend_n signal to the PHY to generate the PHY clock. A New Session Detected interrupt is generated.
5. The application clears the Gate hclk and Stop PHY Clock bits.
6. The function detects USB reset.
7. The function is in normal operating mode

38.12.1.8 Device Mode Session End and SRP With Clock Gating

Sequence of operations:

1. The function detects a USB suspend, and generates a Suspend Detected interrupt. The Host turns off V_{BUS}.
2. The application sets the Stop PHY Clock bit in the Power and Clock Gating Control register, the function asserts the suspend_n signal to the PHY, and the PHY clock stops. The application sets the Gate hclk bit in the Power and Clock Gating Control register, and the function gates the hclk (hclk_ctl) to AHB-domain modules other than the BIU.
3. The function remains in Low-Power mode.
4. The application clears the Gate hclk and Stop PHY Clock bits.
5. The application sets the SRP Request bit, and the function drives data line and V_{BUS} pulsing.
6. The Host turns on V_{BUS}, detects Device connection, and drives a USB reset.
7. The function is in normal operating mode.

38.13 Miscellaneous Topics

38.13.1 Data FIFO RAM Allocation

The External RAM must be allocated among different FIFOs in the function before any transactions can start. The application must follow this procedure every time it changes function FIFO RAM allocation.

The application must allocate data RAM per FIFO based on the AHB's operating frequency, the PHY Clock frequency, the available AHB bandwidth, and the performance required on the USB. Based on the above mentioned criteria, the application must provide a table as described below with RAM sizes for each FIFO in each mode.

38.13.1.1 Device Mode

The following sections cover the programming model for Device mode operation.

Shared Tx FIFO Operation

Considerations for allocating data RAM for some of the FIFOs in Device mode are listed here:

1. Receive FIFO RAM allocation:

- RAM for SETUP Packets: 4 * *n* + 6 locations must be Reserved in the receive FIFO to receive up to *n* SETUP packets on control endpoints, where *n* is the number of control endpoints the Device function supports. The function does not use these locations, which are Reserved for SETUP packets, to write any other data.
- 1 location for Global OUT NAK
- Status information is written to the FIFO along with each received packet. Therefore, a minimum space of (Largest Packet Size/4) + 1 must be allotted to receive packets. If a high-bandwidth Endpoint is enabled, or multiple isochronous endpoints are enabled, then at least two (Largest Packet Size/4) + 1 spaces must be allotted to receive back-to-back packets. Typically, two (Largest Packet Size/4) + 1 spaces are recommended so that when the previous packet is being transferred to AHB, the USB can receive the subsequent packet. If AHB latency is high, you must allocate enough space to receive multiple packets. This is critical to prevent dropping any isochronous packets.
- Along with each Endpoint's last packet, transfer complete status information is also pushed to the FIFO. Typically, one location for each OUT Endpoint is recommended.

2. Transmit FIFO RAM allocation:
   - The RAM size for the Periodic Transmit FIFO must equal the maximum amount of data that can be transmitted in a single microframe. The function does not use any data RAM allocated over this requirement, and when data RAM allocated is less than this requirement, the function can malfunction.
   - The minimum amount of RAM required for the Non-periodic Transmit FIFO is the largest maximum packet size among all supported non-periodic IN endpoints.
   - More space allocated in the Transmit Non-periodic FIFO results in better performance on the USB and can hide AHB latencies. Typically, two Largest Packet Sizes' worth of space is recommended, so that when the current packet is under transfer to the USB, the AHB can get the next packet. If the AHB latency is large, then you must allocate enough space to buffer multiple packets.
   - It is assumed that i number of periodic FIFOs is implemented in Device mode.

FIFO Name	Data RAM Size
Receive Data FIFO	rx_fifo_size. This must include RAM for SETUP packets, OUT Endpoint control information and data OUT packets, as mentioned earlier.
non-periodic Transmit FIFO	tx_fifo_size[0]
Periodic Transmit FIFO 1	tx_fifo_size[1]
Periodic Transmit FIFO 2	tx_fifo_size[2]
...	...
Periodic Transmit FIFO i	tx_fifo_size[i]

With this information at hand, the following registers must be programmed as follows:

1. Receive FIFO Size Register (GRXFSIZ)
   - GRXFSIZ.RxFDep = rx_fifo_size;
2. Non-periodic Transmit FIFO Size Register (GNPTXFSIZ)
   - GNPTXFSIZ.NPTxFDep = tx_fifo_size[0];
   - GNPTXFSIZ.NPTxFStAddr = rx_fifo_size;
3. Device Periodic Transmit FIFO 1 Size Register (DPTXFSIZ1)
   - DPTXFSIZ1.DPTxFStAddr = GNPTXFSIZ.NPTxFStAddr + tx_fifo_size[0];

4. Device Periodic Transmit FIFO 2 Size Register (DPTXFSIZ2)
   - DPTXFSIZ2.DPTxFStAddr = DPTXFSIZ1.DPTxFStAddr + tx_fifo_size[1];
5. Device Periodic Transmit FIFO nSize Register (DPTXFSIZn)
   - DPTXFSIZn.DPTxFStAddr = DPTXFSIZn – 1.DPTxFStAddr + tx_fifo_size[n – 1];
6. The transmit FIFOs and receive FIFO must be flushed after the RAM allocation is done, for proper FIFO function.
   - GRSTCTL.TxFNum = 5'h10
   - GRSTCTL.TxFFlush = 1'b1
   - GRSTCTL.RxFFlush = 1'b1
   - The application must wait until the TxFFlush bit and RxFFlush bits are cleared before performing any operation on the function.

38.13.1.2 Host Mode

FIFO Name	Data RAM Size
Receive Data FIFO	rx_fifo_size
Non-periodic Transmit FIFO	tx_fifo_size[0]
IN Endpoint Transmit FIFO	tx_fifo_size[1]

With this information at hand, the following registers must be programmed as follows:

1. Receive FIFO Size Register (GRXFSIZ)
   - GRXFSIZ. RxFDep = rx_fifo_size;
2. Non-periodic Transmit FIFO Size Register (GNPTXFSIZ)
   - GNPTXFSIZ. NPTxFDe = tx_fifo_size[0];
   - GNPTXFSIZ. NPTxFSt Addr = rx_fifo_size;
3. Host Periodic Transmit FIFO Size Register (HPTXFSIZ)
   - HPTXFSI Z. PTxFSize = tx_fifo_size[1];
   - HPTXFSIZ. PTxFSt Addr = GNPTXFSIZ. NPTxFSt Addr + tx_fifo_size[0];
4. The transmit FIFOs and receive FIFO must be flushed after RAM allocation for proper FIFO function.
   - GRSTCTL. TxFNum = 5' h10
   - GRSTCTL. TxFFlush = 1' b1
   - GRSTCTL. RxFFlush = 1' b1
   - The application must wait until the TxFFlush bit and the RxFFlush bits are cleared before performing any operation on the function.

38.13.2 Dynamic FIFO Allocation

The application can change the RAM allocation for each FIFO during the operation of the function.

Host Mode

In Host mode, before changing FIFO data RAM allocation, the application must determine the following.

- All channels are disabled
- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in *Data FIFO RAM Allocation* on page 1500.

After reallocating the FIFO data RAM, the application must flush all FIFOs in the function using the GRSTCTL.TxFIFO Flush and GRSTCTL.RxFIFO Flush fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation.

Device Mode

In Device mode, before changing FIFO data RAM allocation, the application must determine the following.

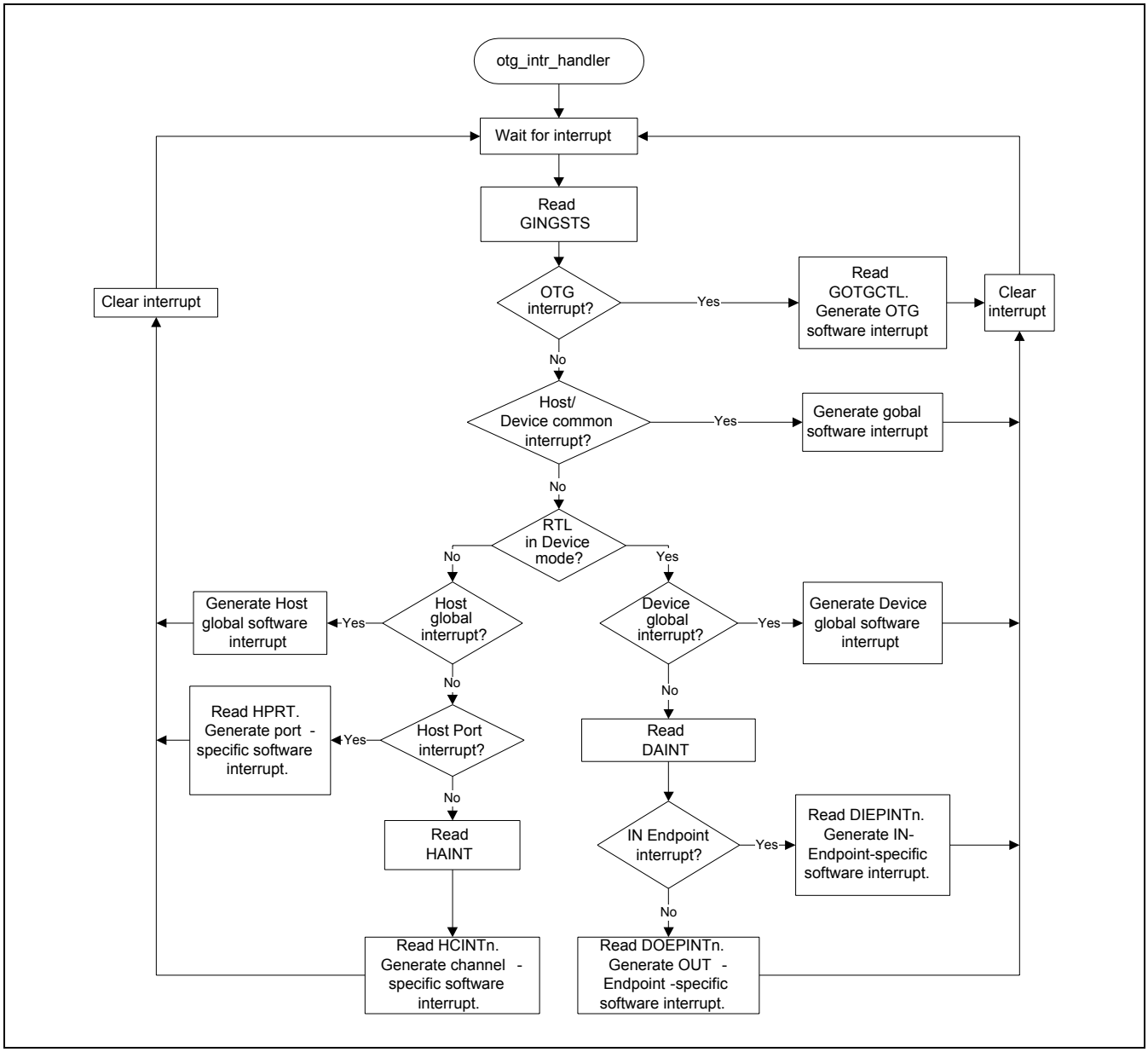
- All IN and OUT endpoints are disabled
- NAK mode is enabled in the function on all IN endpoints
- Global OUT NAK mode is enabled in the function
- All FIFOs are empty

Once these conditions are met, the application can reallocate FIFO data RAM as explained in *Data FIFO RAM Allocation* on page 1500. When NAK mode is enabled in the function, the function responds with a NAK handshake on all tokens received on the USB, except for SETUP packets.

After the reallocating the FIFO data RAM, the application must flush all FIFOs in the function using the GRSTCTL.TxFIFO Flush and GRSTCTL.RxFIFO Flush fields. Flushing is required to reset the pointers in the FIFOs for proper FIFO operation after reallocation.

38.13.3 Function Interrupt Handler

Figure 38-105. Function Interrupt Handler



39. Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) is a full-duplex, synchronous, character-oriented (byte) port that enables the exchange of data with other serial devices. The SPI is master on the serial port supporting a 3-wire interface (receive, transmit, and clock). The SPI is also a slave device to the on-chip peripheral bus (OPB) for communication to the processor.

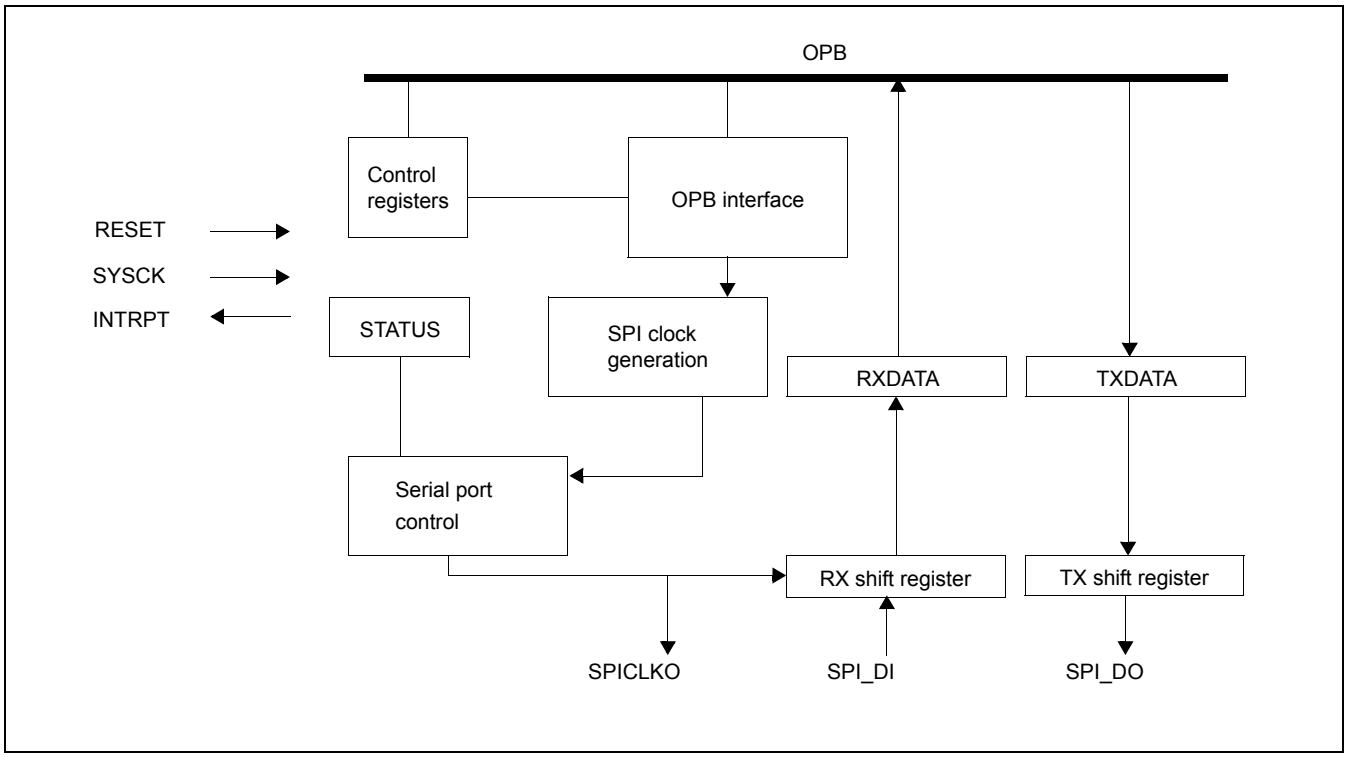
The Serial Peripheral Interface has the following features:

- Three-wire serial port interface
- Full-duplex synchronous operation
- SPI bus master
- OPB bus slave
- Programmable clock rate divider
- Clock inversion
- Optional clock phase
- Reverse data
- Local data loop back for test

39.1 Functional Description

Figure 39-1 shows the functional diagram of the port. The SPI accepts instructions from the OPB to read/write the control registers, or to read/write data from/to the data buffers, RxD and TxD. The SPI can then accept an instruction to initiate a transmit/receive cycle when SPI control register serial data transmit/receive bit is enabled (SPI0_CR[STP] = 1) (see *SPI Control Register (SPI0_CR)* on page 1509). This activity is controlled by the OPB interface circuit.

Figure 39-1. SPI Functional Diagram



If a send/receive cycle is initiated by the OPB, then the serial port control circuit enables the SPI clock for a series of exactly eight clock pulses, shifting eight bits into and eight bits out of the RX and TX shift registers. When this is complete, the SPI clock is again disabled, and an interrupt is generated so the port can be serviced by the interrupt controller. The SPI is always in full duplex mode and always sends and receives one byte of data. The SPI clock is generated internally from the OPB clock (OPBClk) using a clock divider which can be programmed via the SPI0_CDM register (see *SPI Clock Divisor Modulus Register (SPI0_CDM)* on page 1509).

The OPB interface uses 8-bit input and output data buses, referred to as OPB_DBUS[0:7] for input and SL_DBUS[0:7] for output. If an SPI register is addressed for write, then its bits are replaced by the respective bits in the OPB_DBUS. Unused control register bits are reserved and should always be written with 0.

If the SPI is addressed for a register read, then the register bits are placed onto the SL_DBUS and the unused or reserved bits are set to 0. For example, if the OPB reads the SPI0_MODE register, then the SL_DBUS is loaded with 00034567 where 34567 is the content of bits 3, 4, 5, 6, and 7 of the SPI0_MODE register. Reserved bits are set to 0 on a register read.

39.1.1 SPI Interrupt Operation

A single interrupt is generated when the send/receive cycle is complete (RxByteReady). The interrupt is cleared when the received data is read by the OPB. Therefore, even if the received data is not used by the processor, a read operation is still required to clear the interrupt.

39.1.2 Loopback

In loop back operation, the output data is inverted and received into the input receiver.

39.1.3 Typical Operation (Exchange of Data)

The SPI operates as a slave to the OPB. In a typical exchange of data, the processor initiates a data exchange with an external peripheral (an SPI slave). The processor activates the enable signal for that slave using one of the general purpose I/O pins.

To begin the exchange of data, the processor writes the data to be transmitted into the SPI0_TxD register, and then writes the start SPI0_SR[STR] register transmit bit to start the transmission of data. SPI0_SR[STR] is cleared by the SPI after transmission has begun on the serial port.

When the serial port controller recognizes the STR bit, it loads the TxData into the transmit shift register and generates eight clock pulses that simultaneously transmit and receive eight bits of data. The received data is then transferred to the RxData buffer, and an interrupt is issued to the interrupt controller. When the OPB reads the data from the SPI, the interrupt is cleared.

TxData is loaded and SPI0_SR[STR] is cleared when the send/receive cycle is started. Therefore, if SPI0_SR[STR] is written again before this register can be cleared, the new request is lost. If new transmit data is written to the TxData buffer before SPI0_SR[STR] is cleared (that is, old data not yet sent), then the old data in the TxData buffer is overwritten.

39.2 SPI Registers

SPI registers listed on *Table 39-1* are 8-bit wide memory mapped registers. Some registers have reserved bits which should be written as 0, and return 0 when read.

Table 39-1. SPI Registers

Mnemonic	Register	Address	Access	Page
SPI0_MODE	SPI Mode Register	0x4 EF60 0900	R/W	1510
SPI0_RxD	SPI Receive Data Register	0x4 EF60 0901	R	1508
SPI0_TxD	SPI Transmit Data Register	0x4 EF60 0902	R/W	1508
SPI0_CR	SPI Control Register	0x4 EF60 0903	R/W	1509
SPI0_SR	SPI Status Register	0x4 EF60 0904	R	1510
SPI0_CDM	SPI Clock Divisor Modulus Register	0x4 EF60 0906	R/W	1509

39.2.1 SPI Receive Data Register (SPI0_RxD)

The read-only SPI0_RxD register holds data received on the serial port.

Reset value = 0x00

Figure 39-2. SPI Receive Data Register (SPI0_RxD)			
0	Rx0	Receive data 0	
1	Rx1	Receive data 1	
2	Rx2	Receive data 2	
3	Rx3	Receive data 3	
4	Rx4	Receive data 4	
5	Rx5	Receive data 5	
6	Rx6	Receive data 6	
7	Rx7	Receive data 7	

39.2.2 SPI Transmit Data Register (SPI0_TxD)

The read/write SPI0_TxD register holds data transmitted on the serial port.

Reset value = 0x00

Figure 39-3. SPI Transmit Data Register (SPI0_TxD)			
0	Tx0	Transmit data 0	
1	Tx1	Transmit data 1	
2	Tx2	Transmit data 2	
3	Tx3	Transmit data 3	
4	Tx4	Transmit data 4	
5	Tx5	Transmit data 5	
6	Tx5	Transmit data 6	
7	Tx7	Transmit data 7	

39.2.3 SPI Control Register (SPI0_CR)

Figure 39-4 describes the read/write SPI0_CR register. Asserting SPI0_CR[STR] is a request that starts a new transmit/receive cycle on the serial port. When SPI0_CR[STR] = 1, and the SPI is idle, a new byte of data is loaded into the output shift register and a new transmit/receive cycle begins. STR is cleared by the SPI when the transmit/receive begins. The delay between the setting of SPI0_CR[STR] = 1 and the beginning of transmit is variable and depends on the SPI port frequency.

If the port is busy, SPI0_CR[STR] is a pending request and cleared when the next send/receive cycle begins. If SPI0_CR[STR] is asserted twice before it can be cleared, the request is lost.

Note: SPI0_CDM, SPI0_MODE, and SPI0_TxD registers should be written in order to configure the SPI before writing SPI0_CR.

Reset value = 0x00

Figure 39-4. SPI Control Register (SPI0_CR)

0:6		Reserved	
7	STR	Start serial data transmit/receive.	Setting SPI0_CR[STR] = 1 initiates transmit and receive of one character on the SPI. Reset to 0 after serial transmission has begun.

39.2.4 SPI Clock Divisor Modulus Register (SPI0_CDM)

The SPI clock is generated from the OPB clock by a clock divider. The divide ratio is controlled by the value of the SPI0_CDM register, which is initially reset to 0. SPI0_CDM represents a programmable binary number (with bit 0 as the most significant bit (msb)) that is used for the initial value of a count down register. When the count equals zero, the counter is again loaded with the value of SPI0_CDM. This counter is used to generate the output SPI clock (SCPClkOut). The output clock frequency is given by the ratio:

SCPClkOut = OPBCLK/(4(CDM+1)) where CDM = 0–255.

Therefore, the minimum divide ratio is 4 when CDM = 0, and the maximum is 1024 when CDM = 255.

Reset value = 0x00

Figure 39-5. SPI Clock Divisor Modulus Register (SPI0_CDM)

0	CDM0	Clock divisor modulus 0	
1	CDM1	Clock divisor modulus 1	
2	CDM2	Clock divisor modulus 2	
3	CDM3	Clock divisor modulus 3	
4	CDM4	Clock divisor modulus 4	
5	CDM5	Clock divisor modulus 5	
6	CDM6	Clock divisor modulus 6	
7	CDM7	Clock divisor modulus 7	

39.2.5 SPI Status Register (SPI0_SR)

Figure 39-6 describes the SPI0_SR register bits. After a transmit/receive cycle is complete, the received data is loaded into the RxData buffer. The SPI0_SR[RBR] is asserted and this creates an interrupt. If RxData already contained previously received data, then that data is overwritten. SPI0_SR[RBR] also confirms that the transmit byte has been sent. SPI0_SR[BSY] is asserted one cycle after the start of send/receive and de-asserted one cycle after the send/receive is complete.

Reset value = 0x00

Figure 39-6. SPI Status Register (SPI0_SR)

0:5		Reserved	
6	BSY	Busy 0 Serial port is idle. 1 Serial transmit/receive is active.	
7	RBR	RxByteReady 0 RxD buffer is empty. 1 Serial data receive is complete and RxD is available.	

39.2.6 SPI Mode Register (SPI0_MODE)

SPI is configured via SPI0_MODE register as described in *Figure 39-7* (SPI0_MODE[SPE]). If the other bits of SPI0_MODE and SPI0_CDM are changed while the serial port is active, the behavior will not be predictable. When the SPI0_MODE register is at its reset value (00000000), the output clock, SCPClkOut, is forced to 0.

Reset value = 0x00

Figure 39-7. SPI Mode Register (SPI0_MODE)

0:2		Reserved	
3	SCP	Serial Clock Phase 0 Data is latched on trailing edge of clock 1 Data is latched on leading edge of clock	
4	SPE	Serial Port Enabled 0 Serial port disabled 1 Serial port enabled	SPI0_MODE[SPE] is used to enable the SPI port operation. When SPI0_MODE[SPE] = 0 the serial port control circuits are put into a known reset state. When SPI0_MODE[SPE] = 1, the port is enabled and ready to initiate a send/receive cycle. This bit must be asserted before transmission is started (writing to the SPI0_CR register). If SPI0_MODE[SPE] = 0 while the port is transmitting, the current transmit cycle is completed before SPI0_MODE[SPE] = 0 is recognized.
5	RD	Reverse Data 0 Data Bit 0 is transmitted first. 1 Data Bit 7 is transmitted first.	
6	CI	Clock Invert 0 Idle clock = 0, Active clock = 1. 1 Idle clock = 1, Active clock = 0.	
7	IL	Internal Loopback 0 Loopback is disabled. 1 Loopback is enabled.	When SPI0_MODE[IL] = 1 the serial output data is inverted and connected to the receiver input.

The serial port is synchronous to the port clock, SPIClkOut. Data is driven on the link at one edge of the clock (usually the leading positive edge) and data is captured at the other edge (usually the trailing falling edge).

Because the clock is generated by the SPI master and sent to the slave (on another chip), the master sees the leading edge long before the slave. The slave must wait for the SPIClkOut leading edge to propagate from the master to the slave, then assert its data to serial data out. The master then waits for the slave to respond and for the slave's data to propagate back to the master. Finally, on the trailing edge of the clock the data is captured first by the master, and then by the slave when it sees the trailing edge of the clock.

Sufficient time must be allowed for round trip delay between master and slave when choosing a serial port clock frequency in order to guarantee that data is stable before capturing it at both master and slave.

There are four SPI bus modes which are controlled with Clock Invert, SPIO_MODE[CI], and Serial Clock Phase, SPIO_MODE[SCP]. Data transfer with each of these modes is illustrated below.

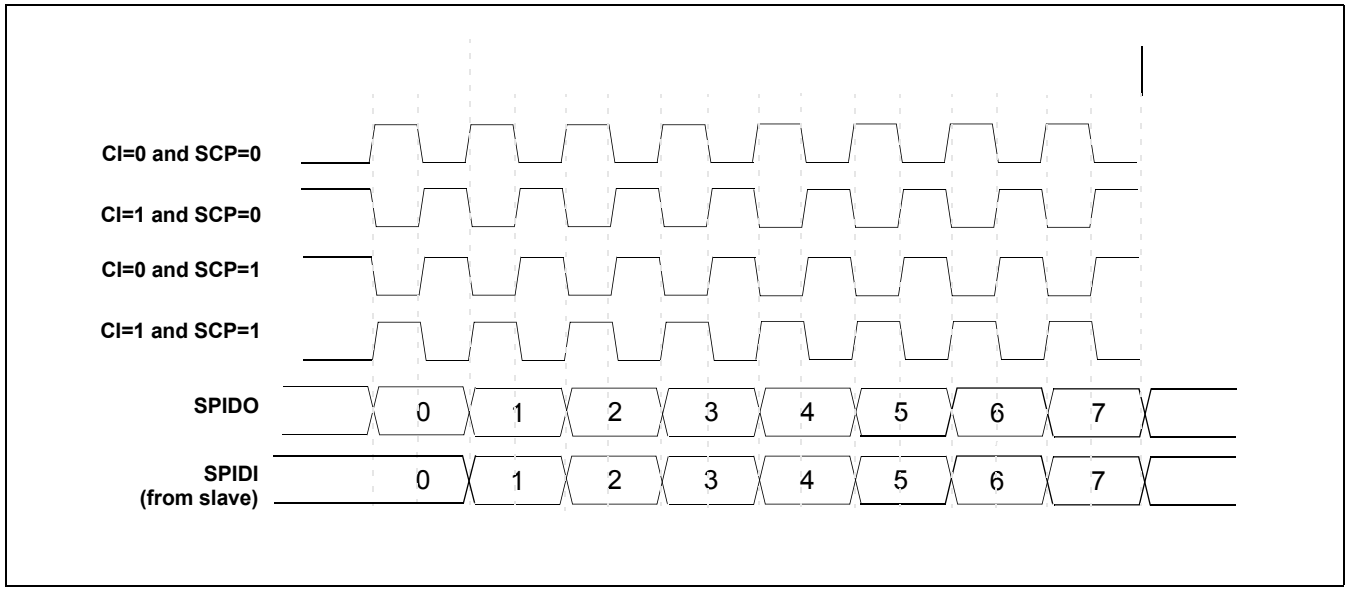
With SPIO_MODE[SCP] = 0 and SPIO_MODE[CI] = 0, data is driven on the leading/rising edge of the serial clock and latched on the trailing/falling edge.

With SPIO_MODE[SCP] = 1 and SPIO_MODE[CI] = 0, data is latched on the leading/rising edge of the serial clock and then asserted on the trailing/falling edge. In this case, data from the slave device and the master device must already be on the bus before the first bit is latched.

Table 39-2. Data Driven or Latched Relative to SPIClkOut Clock Edge

SPIO_MODE[SCP]	SPIO_MODE[CI]	Data Drive Relative to SPICClkOut	Data Latched to SPICClkOutr
0	0	Rising Edge	Falling Edge
0	1	Falling Edge	Rising Edge
1	0	Falling Edge	Rising Edge
1	1	Rising Edge	Falling Edge

Figure 39-8. SPI Data Relative To SPIClkOut (SPIO_MODE[SCP], SPIO_MODE[CI])



Part V. Reference

40. Instruction Set

All instruction processing in the PPC460EX/EXr/GT is handled by the PPC440 processor. Refer to the *PPC440H6 Processor User's Manual* for details.

41. Floating Point Instruction Set

Descriptions of the PPC460EX/EXr/GT floating point instructions follow. Each description contains the following elements:

- Instruction names (mnemonic and full)
- Instruction syntax
- Instruction format diagram
- Pseudocode description
- Prose description
- Registers altered

Where appropriate, instruction descriptions list exceptions and invalid instruction forms, and provide programming notes.

Table 41-1 summarizes the PPC460EX/EXr/GT instruction set by category.

Table 41-1. Instruction Categories

Category	Subcategory	Instruction Types
Floating Point	Computational	Elementary arithmetic, multiply-add, rounding and converting, square root and reciprocal estimate
	Noncomputational	Load/store, move, compare, Floating-Point Status and Control Register

41.1 Instruction Set Portability

To support embedded real-time applications, the PPC460EX/EXr/GT implements the defined floating point instruction set of the Book-E Enhanced PowerPC Architecture, with the exception the **fctid**, **fsqrt**, and **fsqrts** instructions.

The Book-E Enhanced PowerPC Architecture defines some instructions that have record forms, often called "dot forms,' that update the CR1 field of the Condition Register (CR). The record forms of these instructions are not implemented.

41.2 Instruction Formats

For more detailed information about instruction formats, including a summary of instruction field usage and instruction format diagrams for the PPC460EX/EXr/GT, see *Floating Point Instruction Summary* on page 1579.

Instructions are four bytes long. Instruction addresses are always word-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions also have an extended opcode field. The remaining instruction bits are contained in additional fields. All instruction fields belong to one of the following categories:

- Defined

  These instruction fields contain values, such as opcodes, that cannot be altered. The instruction format diagrams specify the values of defined fields.

- Variable

  These fields contain operands, such as general purpose register specifiers and immediate values, each of which may contain any one of a number of values. The instruction format diagrams specify the field names of variable fields.

- Reserved

  Bits in a reserved field should be set to 0. In the instruction format diagrams, reserved fields are shaded.

If any bit in a defined field does not contain the specified value, the instruction is illegal and an Illegal Instruction exception type Program interrupt occurs. If any bit in a reserved field does not contain 0, the instruction form is invalid and its result is architecturally undefined. Unless otherwise noted, the PPC460EX/EXr/GT executes all invalid instruction forms without causing an Illegal Instruction exception.

41.3 Pseudocode

The pseudocode that appears in the instruction descriptions provides a semi-formal language for describing instruction operations.

The pseudocode uses the following notation:

+	Twos complement addition
%	Remainder of an integer division; (33 % 32) = 1.
$<^{u}$, $>^{u}$	Unsigned comparison relations
(FPR(r))	The contents of FPR r, where $0 \le r \le 31$.
(FRx)	The contents of an FPR, where *x* is A, B, C, S, or T
(GPR(r))	The contents of GPR r, where $0 \le r \le 31$.
(RA|0)	The contents of the register RA or 0, if the RA field is 0.
(Rx)	The contents of a GPR, where *x* is A, B, S, or T
0bn	A binary number
0xn	A hexadecimal number
<, >	Signed comparison relations
=	Assignment
=, ≠	Equal, not equal relations
CEIL(x)	Least integer ≥ x.
CIA	Current instruction address; the 32-bit address of the instruction being described by a sequence of pseudocode. This address is used to set the next instruction address (NIA). Does not correspond to any architected register.
DCR(DCRN)	A Device Control Register (DCR) specified by the DCRF field in an **mfdcr** or **mtdcr** instruction
EA	Effective address; the 32-bit address, derived by applying indexing or indirect addressing rules to the specified operand, that specifies an location in main storage.
EXTS(x)	The result of extending *x* on the left with sign bits.
FLD	An instruction or register field
FLD_b	A bit in a named instruction or register field
$FLD_{b,b,\ldots}$	A list of bits, by number or name, in a named instruction or register field

$FLD_{b:b}$	A range of bits in a named instruction or register field
FRx	An FPR, where *x* is A, B, C, S, or T
GPR(r)	General Purpose Register (GPR) r, where $0 \le r \le 31$.
GPRs	RA, RB, . . .
MASK(MB,ME)	Mask having 1s in positions MB through ME (wrapping if MB > ME) and 0s elsewhere.
MS(addr, n)	The number of bytes represented by *n* at the location in main storage represented by *addr*.
NIA	Next instruction address; the 32-bit address of the next instruction to be executed. In pseudo-code, a successful branch is indicated by assigning a value to NIA. For instructions that do not branch, the NIA is CIA +4.
PC	Program counter.
Rx	A GPR, where *x* is A, B, S, or T
REG[FLD, FLD . . .]	A list of fields in a named register
REG[FLD:FLD]	A range of fields in a named register
REG[FLD]	A field in a named register
REG_b	A bit in a named register
$REG_{b,b,\ldots}$	A list of bits, by number or name, in a named register
$REG_{b:b}$	A range of bits in a named register
RESERVE	Reserve bit; indicates whether a process has reserved a block of storage.
ROTL((RS),n)	Rotate left; the contents of RS are shifted left the number of bits specified by *n*.
SPR(SPRN)	A Special Purpose Register (SPR) specified by the SPRF field in an **mfspr** or **mtspr** instruction
$c_{0:3}$	A four-bit object used to store condition results in compare instructions.
do	Do loop. "to" and "by" clauses specify incrementing an iteration variable; "while" and "until" clauses specify terminating conditions. Indenting indicates the scope of a loop.
if...then...else...	Conditional execution; if *condition* then *a* else *b*, where *a* and *b* represent one or more pseudocode statements. Indenting indicates the ranges of *a* and *b*. If *b* is null, the else does not appear.
instruction(EA)	An instruction operating on a data or instruction cache block associated with an EA.
leave	Leave innermost do loop or do loop specified in a leave statement.
n	A decimal number
^nb	The bit or bit value *b* is replicated *n* times.
xx	Bit positions which are don't-cares.
$\|\|$	Concatenation
$\times$	Multiplication
$\div$	Division yielding a quotient
$\oplus$	Exclusive-OR (XOR) logical operator
–	Twos complement subtraction, unary minus
$\neg$	NOT logical operator
$\wedge$	AND logical operator
$\vee$	OR logical operator

41.3.1 Operator Precedence

Table 41-2 lists the pseudocode operators and their associativity in descending order of precedence:

Table 41-2. Operator Precedence

Operators	Associativity
REG_b, REG[FLD], function evaluation	Left to right
^{n}b	Right to left
¬, – (unary minus)	Right to left
×, ÷	Left to right
+, –	Left to right
||	Left to right
=, ≠, <, >, $\overset{u}{<}$, $\overset{u}{>}$	Left to right
∧, ⊕	Left to right
∨	Left to right
←	None

41.4 Register Usage

Each instruction description lists the registers altered by the instruction. Some register changes are explicitly detailed in the instruction description (for example, the target register of a load instruction). Some instructions also change other registers, but the details of the changes are not included in the instruction descriptions. Common examples of these kinds of register changes include the Floating-Point Registers (FPRs) and the Floating-Point Status and Control Register (FPSCR). For discussion of the FPRs, see *Floating-Point Registers (FPR0:31)* on page 160. For discussion of the FPSCR, see *Floating-Point Status and Control Register (FPSCR)* on page 161.

41.5 Floating-Point Instructions

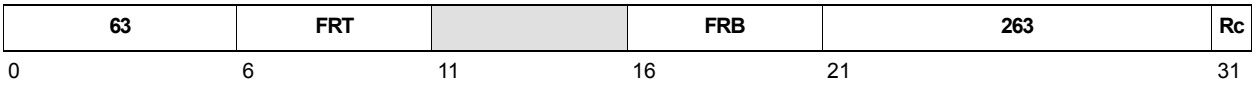
Primary opcode 63 is used for the double-precision arithmetic instructions and miscellaneous instructions (for example, the *Floating-Point Status and Control Register Manipulation* instructions). Primary opcode 59 is used for the single-precision arithmetic instructions.

The single-precision instructions for which there is a corresponding double-precision instruction have the same format and extended opcode as that double-precision instruction.

41.6 Alphabetical Instruction Listing

The following pages list the defined floating point instructions implemented in the PPC460EX/EXr/GT.

fabs FRT, FRB Rc = 0



$(FRT) \leftarrow 0 \,||\, (FRB)_{1:63}$

The contents of FRB, with bit 0 set to zero, are placed into FRT.

Registers Altered

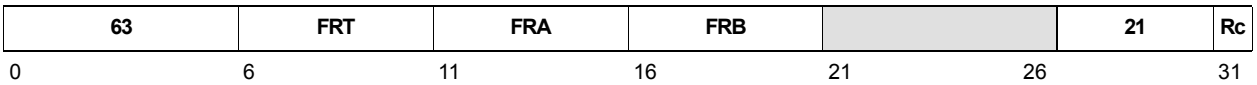
- FRT

Exceptions

An attempt to execute **fabs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fadd FRT, FRA, FRB Rc = 0



(FRT) ← (FRA) + (FRB)

The floating-point operand in FRA is added to the floating-point operand in FRB.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of the FPSCR[RN] and placed into FRT.

Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added or subtracted as appropriate, depending on the signs of the operands, to form an intermediate sum. All 53 bits of the significand and the three guard bits (G, R, X) enter into the computation.

If a carry occurs, the significand of the sum is shifted right one bit position and the exponent is increased by one.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

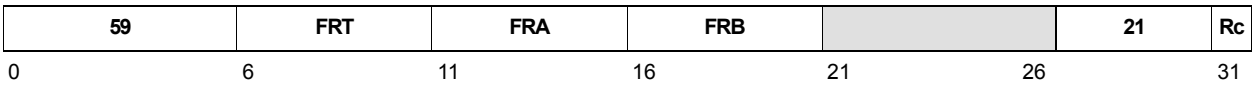
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI]

Exceptions

An attempt to execute **fadd** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fadds FRT, FRA, FRB Rc = 0



(FRT) ← (FRA) $+_{sp}$ (FRB)

The floating-point operand in FRA is added to the floating-point operand in FRB.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of the FPSCR[RN] and placed into FRT.

Floating-point addition is based on exponent comparison and addition of the two significands. The exponents of the two operands are compared, and the significand accompanying the smaller exponent is shifted right, with its exponent increased by one for each bit shifted, until the two exponents are equal. The two significands are then added or subtracted as appropriate, depending on the signs of the operands, to form an intermediate sum. All 53 bits of the significand and the three guard bits (G, R, X) enter into the computation.

If a carry occurs, the significand of the sum is shifted right one bit position and the exponent is increased by one.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

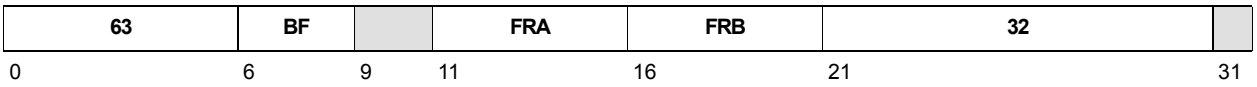
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI]

Exceptions

An attempt to execute **fadds** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fcmpo BF, FRA, FRB



if (FRA) is a NaN or
(FRB) is a NaN then c ← 0b0001
else if (FRA) < (FRB) then c ← 0b1000
else if (FRA) > (FRB) then c ← 0b0100
else c← 0b0010
FPSCR[FPCC] ← c
$CR_{4 \times BF:4 \times BF + 3}$ ← c
if (FRA) is a SNaN or (FRB) is a SNaN then do
    if FPSCR[VE] = 0 then FPSCR[VXVC] ← 1
else if (FRA) is a QNaN or (FRB) is a QNaN then FPSCR[VXVC] ← 1

The floating-point operand in FRA is compared to the floating-point operand in FRB. The result of the compare is placed into the CR field BF and FPSCR[FPCC]. The comparison ignores the sign of zero (that is, +0 is considered equal to –0).

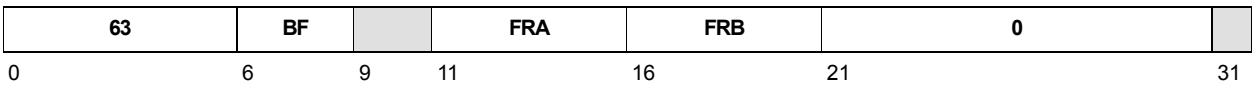
If (FRA) or (FRB) is a NaN, either quiet or signaling, the CR field BF and FPSCR[FPCC] are set to reflect unordered.

If (FRA) or (FRB) is a Signaling NaN and Invalid Operation is disabled (FPSCR[VE] = 0), FPSCR[VXVC] = 1. If neither (FRA) nor (FRB) is a Signaling NaN, but at least one operand is a Quiet NaN, FPSCR[VXVC] = 1.

Registers Altered

- CR[BF]
- FPSCR[FPCC, FX, VXSNAN, VXVC]

fcmpu BF, FRA, FRB



if (FRA) is a NaN or
(FRB) is a NaN then c ← 0b0001
else if (FRA) < (FRB) then c ← 0b1000
else if (FRA) > (FRB) then c ← 0b0100
else c← 0b0010
FPSCR[FPCC] ← c
$CR_{4 \times BF:4 \times BF+3}$ ← c
if ((FRA) is a SNaN or (FRB) is a SNaN) then FPSCR[VXSNAN] ← 1

The floating-point operand in FRA is compared to the floating-point operand in FRB. The result of the compare is placed into the CR field BF and FPSCR[FPCC]. The comparison ignores the sign of zero (that is, +0 is considered equal to –0).

If (FRA) or (FRB) is a NaN, either quiet or signaling, the CR field BF and FPSCR[FPCC] are set to reflect unordered.

If either (FRA) or (FRB) is a Signaling NaN, FPSCR[VXSNAN] = 1.

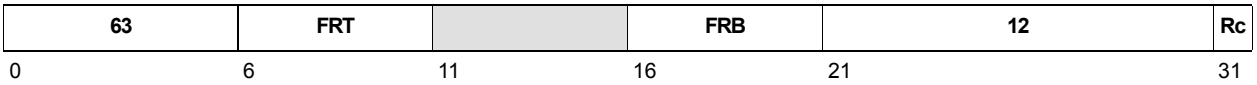
Registers Altered

- CR
- FPSCR[FPCC, FX, VXSNAN]

fctiw

Floating Convert to Integer Word

fctiw FRT, FRB Rc = 0



The floating-point operand in FRB is converted to a 32-bit signed integer, using the rounding mode specified by FPSCR[RN], and placed into $FRT_{32:63}$. $FRT_{0:31}$ are undefined.

If (FRB) is greater than $2^{31}-1$, $FRT_{32:63}$ are set to 0x7FFFFFFF. If (FRB) is less than -2^{31}, $FRT_{32:63}$ are set to 0x80000000.

Except for enabled Invalid Operation Exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.

Registers Altered

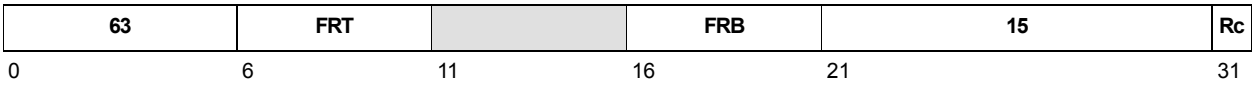
- FRT
- FPSCR[FR, FI, FX, XX, VXSNAN, VXCVI]
- FPSCR[FPRF] undefined

Exceptions

An attempt to execute **fctiw** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fctiwz FRT, FRB Rc = 0



The floating-point operand in FRB is converted to a 32-bit signed integer, using the *Round toward Zero* rounding mode, and placed into $FRT_{32:63}$. $FRT_{0:31}$ are undefined.

If (FRB) is greater than 2^{31}–1, $FRT_{32:63}$ are set to 0x7FFF FFFF. If (FRB) is less than -2^{31}, $FRT_{32:63}$ are set to 0x8000 0000.

Except for enabled Invalid Operation Exceptions, FPSCR[FPRF] is undefined. FPSCR[FR] is set if the result is incremented when rounded. FPSCR[FI] is set if the result is inexact.

Registers Altered

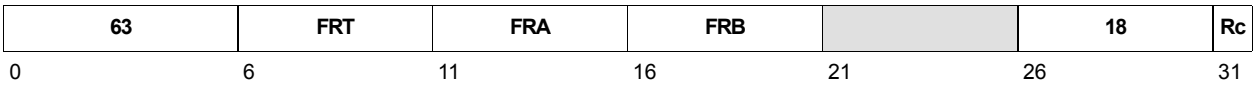
- FRT
- FPSCR[FR, FI, FX, XX, VXSNAN, VXCVI]
- FPSCR[FPRF] undefined

Exceptions

An attempt to execute **fctiwz** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fdiv FRT, FRA, FRB Rc = 0



(FRT) ← (FRA) ÷ (FRB)

The floating-point operand in FRA is divided by the floating-point operand in FRB.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of FPSCR[RN] and placed into FRT.

The remainder is not supplied as a result.

Floating-point division is based on exponent subtraction and division of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1 and Zero Divide Exceptions when FPSCR[ZE] = 1.

Registers Altered

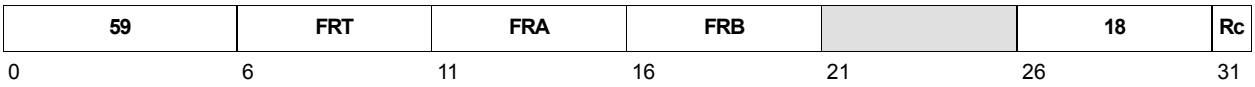
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ]

Exceptions

An attempt to execute **fdiv** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fdivs FRT, FRA, FRB Rc = 0



FRT ← (FRA) ÷ (FRB)

The floating-point operand in FRA is divided by the floating-point operand in FRB.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of FPSCR[RN] and placed into (FRT).

The remainder is not supplied as a result.

Floating-point division is based on exponent subtraction and division of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1 and Zero Divide Exceptions when FPSCR[ZE] = 1.

Registers Altered

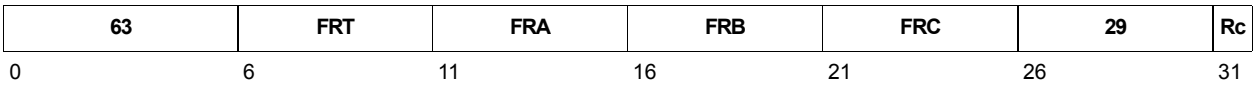
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ]

Exceptions

An attempt to execute **fdivs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fmadd FRT, FRA, FRC, FRB Rc = 0



(FRT) ← [(FRA) × (FRC)] + (FRB)

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is added to this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of the FPSCR[RN] and placed into FRT.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

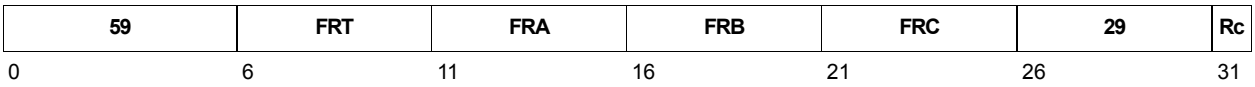
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

An attempt to execute **fmadd** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fmadds FRT, FRA, FRC, FRB Rc = 0



(FRT) ← [(FRA) × (FRC)] + (FRB)

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is added to this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of the FPSCR[RN] and placed into FRT.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

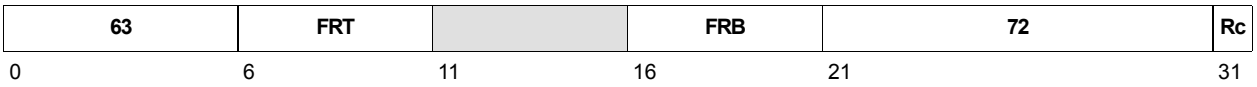
An attempt to execute **fmadds** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fmr

Floating Move Register

fmr FRT, FRB Rc = 0



(FRT) ← FR(FRB)

The contents of FRB are placed into FRT.

Registers Altered

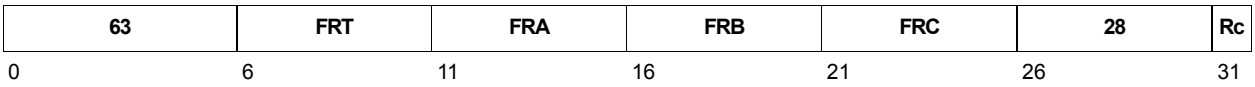
- FRT

Exceptions

An attempt to execute **fmr** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fmsub FRT, FRA, FRC, FRB Rc = 0



(FRT) ← [(FRA) × (FRC)] – (FRB)

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is subtracted from this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of FPSCR[RN] and placed into FRT.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

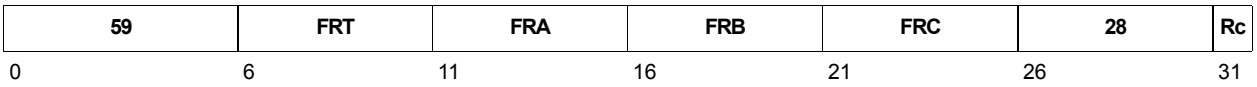
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

An attempt to execute **fmsub** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fmsubs FRT, FRA, FRC, FRB Rc = 0



(FRT) ← [(FRA) × (FRC)] – (FRB)

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is subtracted from this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of FPSCR[RN] and placed into FRT.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation exceptions when FPSCR[VE] = 1.

Registers Altered

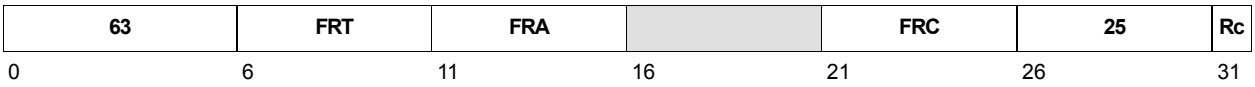
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

An attempt to execute **fmsubs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fmul FRT, FRA, FRC Rc = 0



(FRT) ← (FRA) × (FRC)

The floating-point operand in FRA is multiplied by the floating-point operand in FRC.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of FPSCR[RN] and placed into FRT.

Floating-point multiplication is based on exponent addition and multiplication of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation exceptions when FPSCR[VE] = 1.

Registers Altered

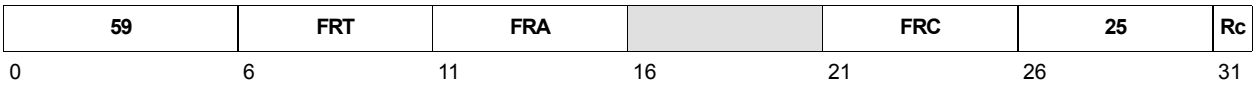
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXIMZ]

Exceptions

An attempt to execute **fmul** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fmuls FRT, FRA, FRC Rc = 0



(FRT) ← (FRA) × (FRC)

The floating-point operand in FRA is multiplied by the floating-point operand in FRC.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of FPSCR[RN] and placed into FRT.

Floating-point multiplication is based on exponent addition and multiplication of the significands.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation exceptions when FPSCR[VE] = 1.

Registers Altered

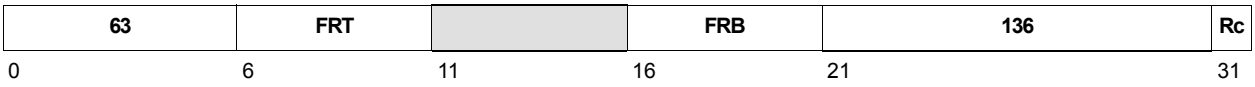
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXIMZ]

Exceptions

An attempt to execute **fmuls** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fnabs FRT, FRB Rc = 0



$(FRT) \leftarrow 1 \,||\, (FRB)_{1:63}$

The contents of FRB, with bit 0 set to one, are placed into FRT.

Registers Altered

- FRT

Exceptions

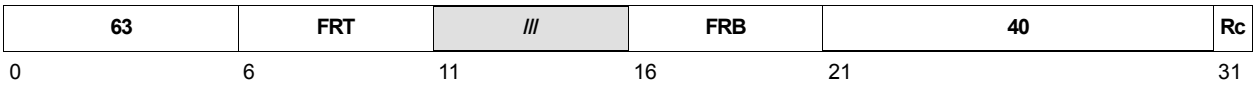
An attempt to execute **fnabs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fneg

Floating Negate

fneg FRT, FRB Rc = 0



$(FRT) \leftarrow \neg(FRB)_0 \,||\, (FRB)_{1:63}$

The contents of FRB, with bit 0 inverted, are placed into FRT.

Registers Altered

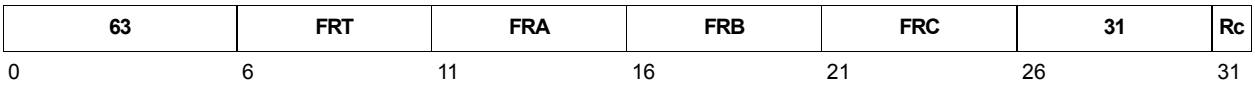
- FRT

Exceptions

An attempt to execute **fneg** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fnmadd FRT, FRA, FRC, FRB Rc = 0



FRT ← –([(FRA) × (FRC)] + (FRB))

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is added to this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of the FPSCR[RN], then negated and placed into (FRT).

This instruction produces the same result as would be obtained by using the **fmadd** instruction and then negating the result, with the following exceptions.

- Quiet NaNs propagate with no effect on their "sign" (high-order) bit.
- Quiet NaNs that are generated as the result of a disabled Invalid Operation Exception have a "sign" bit of 0.
- Signaling NaNs that are converted to Quiet NaNs as the result of a disabled Invalid Operation Exception retain the "sign" bit of the Signaling NaN.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

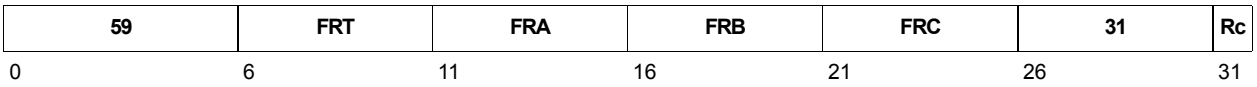
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

An attempt to execute **fnmadd** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fnmadds FRT, FRA, FRC, FRB Rc = 0



(FRT) ← –([(FRA) × (FRC)] + (FRB))

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is added to this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of the FPSCR[RN], then negated and placed into FRT.

This instruction produces the same result as would be obtained by using the **fmadds** instruction and then negating the result, with the following exceptions.

- Quiet NaNs propagate with no effect on their "sign" (high-order) bit.
- Quiet NaNs that are generated as the result of a disabled Invalid Operation Exception have a "sign" bit of 0.
- Signaling NaNs that are converted to Quiet NaNs as the result of a disabled Invalid Operation Exception retain the "sign" bit of the Signaling NaN.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

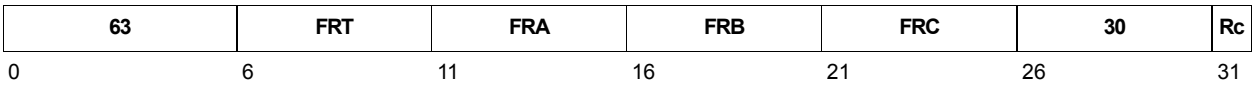
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

An attempt to execute **fnmadds** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fnmsub FRT, FRA, FRC, FRB Rc = 0



(FRT) ← –([(FRA) × (FRC)] – (FRB))

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is subtracted from this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of the FPSCR[RN], then negated and placed into FRT.

This instruction produces the same result as would be obtained by using the **fmsub** instruction and then negating the result, with the following exceptions.

- Quiet NaNs propagate with no effect on their ”sign” (high-order) bit.
- Quiet NaNs that are generated as the result of a disabled Invalid Operation Exception have a “sign” bit of 0.
- Signaling NaNs that are converted to Quiet NaNs as the result of a disabled Invalid Operation Exception retain the “sign” bit of the Signaling NaN.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

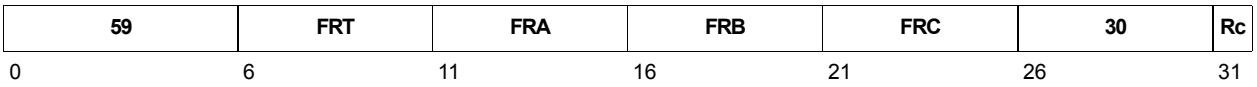
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

An attempt to execute **fnmsub** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fnmsubs FRT, FRA, FRC, FRB Rc = 0



(FRT) ← –([(FRA) × (FRC)] – (FRB))

The floating-point operand in FRA is multiplied by the floating-point operand in FRC. The floating-point operand in FRB is subtracted from this intermediate result.

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of the FPSCR[RN], then negated and placed into FRT.

This instruction produces the same result as would be obtained by using the **fmsubs** instruction and then negating the result, with the following exceptions.

- Quiet NaNs propagate with no effect on their ”sign” (high-order) bit.
- Quiet NaNs that are generated as the result of a disabled Invalid Operation Exception have a “sign” bit of 0.
- Signaling NaNs that are converted to Quiet NaNs as the result of a disabled Invalid Operation Exception retain the “sign” bit of the Signaling NaN.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

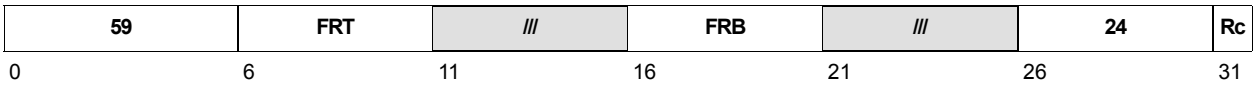
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]

Exceptions

An attempt to execute **fnmsubs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fres FRT, FRB (Rc=0)



(FRT) ← FPReciprocalEstimate(FRB)

A single-precision estimate of the reciprocal of the floating-point operand in FRB is placed into FRT. The estimate is correct to a precision of one part in 2^{13} of the reciprocal of (FRB), that is,

$$\left| \frac{\text{estimate} - \frac{1}{x}}{\frac{1}{x}} \right| \leq \frac{1}{2}13$$

where *x* is the initial value of (FRB).

Table 41-3 summarizes operation with operands having various special values.

*Table 41-3. **fres** Operation with Special Operand Values*

Operand	Result	Exception
$-\infty$	-0	—
-0	$-\infty$[1]	ZX
+0	$+\infty$[1]	ZX
$+\infty$	+0	—
SNaN	QNaN[2]	VXSNAN
QNaN	QNaN	—

Note: 1. No result if FPSCR[ZE] = 1
2. No result if FPSCR[VE] = 1.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1 and Zero Divide Exceptions when FPSCR[ZE] = 1.

Registers Altered

- FRT
- FPSCR[FX, OX, UX, ZX, VXSNAN]
- FPSCR[FPRF, FR, FI] undefined

Exceptions

An attempt to execute **fres** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Architecture Notes

The value placed into FRT may vary between implementations, and between different executions on the same implementation.

frsp

Floating Round to Single Precision

frsp FRT, FRB (Rc=0)

The floating-point operand in FRB is rounded to single-precision, using the rounding mode specified by FPSCR[RN], and placed into FRT.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

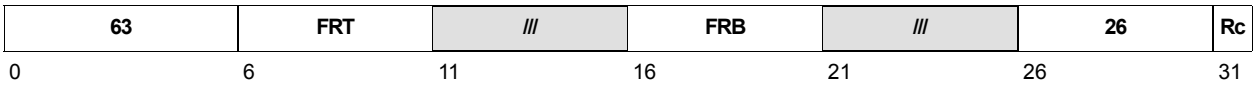
- FRT
- FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN]

Exceptions

An attempt to execute **frsp** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

frsqrte FRT, FRB (Rc=0)



(FRT) ← FPReciprocalSquareRootEstimate(FRB)

A double-precision estimate of the reciprocal of the square root of the floating-point operand in FRB is placed into FRT. The estimate placed is correct to a precision of one part in 2^{13} of the reciprocal of the square root of (FRB), that is,

$$\left| \frac{\left(\text{estimate} - \frac{1}{\sqrt{x}}\right)}{\frac{1}{\sqrt{x}}} \right| \leq \frac{1}{2}13$$

where *x* is the initial value in FPR(FRB). Note that the value placed into FPR(FRT) may vary between implementations, and between different executions on the same implementation.

Table 41-4 summarizes operation with various special values of the operand.

*Table 41-4. **frsqrte** Operation with Special Operand Values*

Operand	Result	Exception
–∞	QNaN[2]	VXSQRT
< 0	QNaN[2]	VXSQRT
–0	–∞[1]	ZX
+0	+∞[1]	ZX
+∞	+0	—
SNaN	QNaN[2]	VXSNAN
QNaN	QNaN	—

Note: 1. No result if FPSCR[ZE] = 1
2. No result if FPSCR[VE] = 1.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1 and Zero Divide Exceptions when FPSCR[ZE] = 1.

Registers Altered

- FRT
- FPSCR[FX, ZX, VXSNAN, VXSQRT]
- FPSCR[FPRF, FR, FI] undefined

Exceptions

An attempt to execute **frsqrte** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

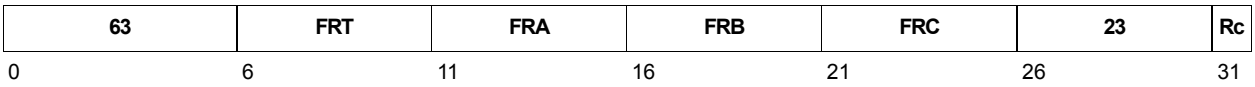
Architecture Notes

No single-precision version of this instruction is provided. If the floating-point operand in FRB is representable in single format, so is the resulting value of FRT.

Implementation Note

The precision of this estimate in this implementation is higher than that specified in Book-E (2^{-5}). Therefore, code relying on this higher precision may not work on other PowerPC implementations.

fsel FRT, FRA, FRC, FRB (Rc=0)



if (FRA) ≥ 0.0 then (FRT) ← (FRC)

else (FRT) ← (FRB)

The floating-point operand in FRA is compared to zero. If the operand is greater than or equal to 0, FRT is set to the contents of FRC. If the operand is less than 0 or is a NaN, FRT is set to the contents of FRB. The comparison ignores the sign of zero (that is, +0 is considered equal to –0).

Registers Altered

- FRT

Exceptions

An attempt to execute **fsel[.]** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Programming Notes

Care must be taken in using fsel if compatibility with IEEE 754 is required, or if the values being tested can be NaNs or infinities.

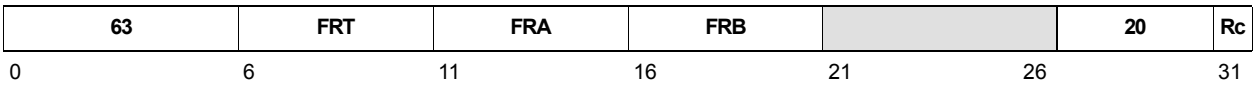
Architecture Notes

The *Select* instruction is similar to a *Move* instruction, and therefore does not alter FPSCR.

fsub

Floating Subtract

fsub FRT, FRA, FRB Rc = 0



(FRT) ← (FRA) – (FRB)

The floating-point operand in FPR(FRB) is subtracted from the floating-point operand in FPR(FRA).

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to double precision under control of FPSCR[RN] and placed into FRT.

The operation of **fsub** is identical to that of **fadd**, except that the contents of FRB participate in the operation with the sign bit (bit 0) inverted.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

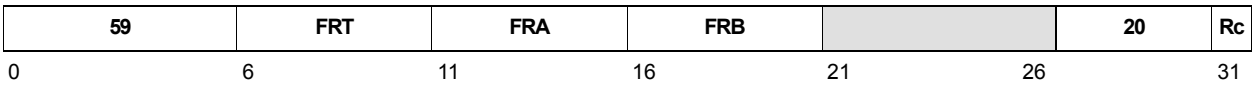
- FRT
- FPSRP[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN]

Exceptions

An attempt to execute **fsub** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

fsubs FRT, FRA, FRB Rc = 0



(FRT) ← (FRA) – (FRB)

The floating-point operand in FPR(FRB) is subtracted from the floating-point operand in FPR(FRA).

If the most significant bit of the resultant significand is not 1, the result is normalized. The result is rounded to single precision under control of FPSCR[RN] and placed into FRT.

The operation of **fsubs** is identical to that of **fadds**, except that the contents of FRB participate in the operation with the sign bit (bit 0) inverted.

FPSCR[FPRF] is set to the class and sign of the result, except for Invalid Operation Exceptions when FPSCR[VE] = 1.

Registers Altered

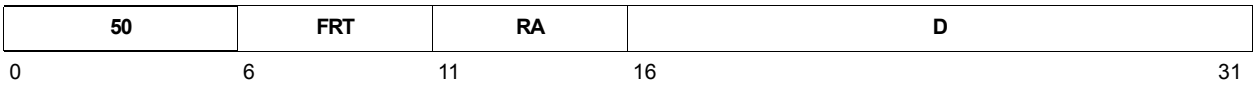
- FRT
- FPSCR[FPRF, FR, FI FX, OX, UX, XX, VXSNAN]

Exceptions

An attempt to execute **fsubs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

lfd FRT, D(RA)



EA ← (RA|0) + EXTS(D)
(FRT) ← MEM(EA,8)

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D field to 32 bits.

The doubleword in storage addressed by EA is placed into FRT.

Registers Altered

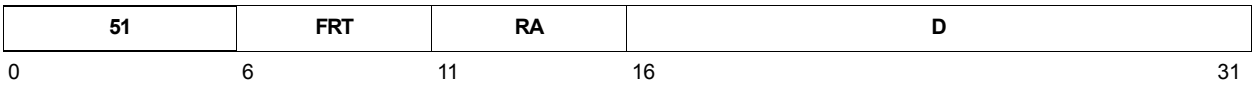
- FRT

Exceptions

An attempt to execute **lfd** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

lfdu FRT, D(RA)



EA ← (RA|0) + EXTS(D)
(FRT) ← MEM(EA,8)
RA ← EA

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D field to 32 bits.

The doubleword in storage addressed by EA is placed into FRT.

EA is placed into RA.

Registers Altered

- FRT
- RA

Exceptions

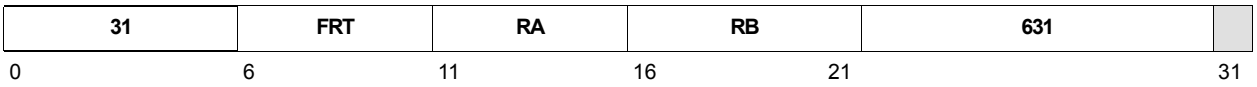
An attempt to execute **lfdu** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

Invalid Forms

RA = 0

lfdux FRT, RA, RB



EA ← (RA|0) + (RB)
(FRT) ← MEM(EA,8)
RA ← EA

Let the effective address (EA) be the sum (RA|0) and the index specified by (RB).

The doubleword in storage addressed by EA is placed into FRT.

EA is placed into RA.

Registers Altered

- FRT
- RA

Exceptions

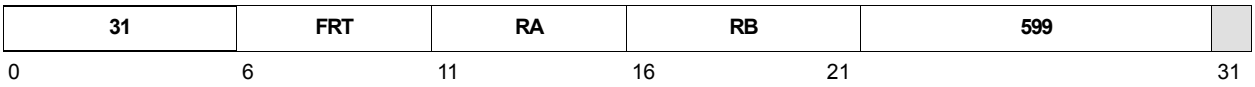
An attempt to execute **lfdux** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

Invalid Forms

RA = 0

lfdx FRT, RA, RB



EA ← (RA|0) + (RB)
(FRT) ← MEM(EA,8)

Let the effective address (EA) be the sum (RA|0) and the index specified by (RB).

The doubleword in storage addressed by EA is placed into FRT.

Registers Altered

- FRT

Exceptions

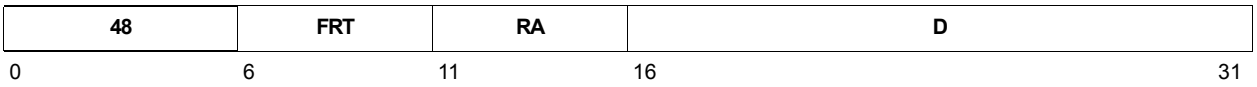
An attempt to execute **lfdx** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

lfs FRT, D(RA)



EA ← (RA|0) + EXTS(D)
(FRT) ← DOUBLE(MEM(EA,4))

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D field to 32 bits.

The word in storage addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double format (see *Data Handling and Precision* on page 168) and placed into FRT.

Registers Altered

- FRT

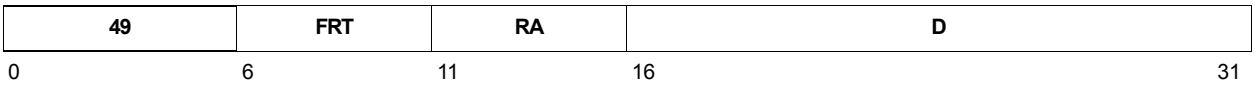
Exceptions

An attempt to execute **lfs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

lfsu FRT, D(RA)



EA ← (RA|0) + EXTS(D)
(FRT) ← DOUBLE(MEM(EA,4))
RA ← EA

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D field to 32 bits.

The word in storage addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double format (see *Data Handling and Precision* on page 168) and placed into FRT.

EA is placed into RA.

Registers Altered

- FRT
- RA

Exceptions

An attempt to execute **lfsu** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

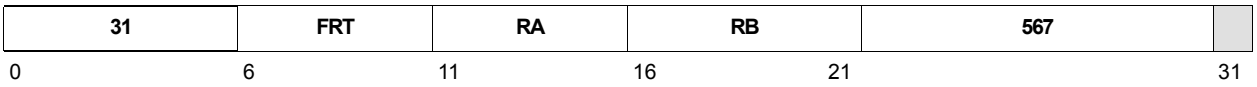
Invalid Forms

RA = 0

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

lfsux FRT, RA, RB



EA ← (RA|0) + (RB)
(FRT) ← DOUBLE(MEM(EA,4))
RA ← EA

Let the effective address (EA) be the sum (RA|0) and the index specified by (RB).

The word in storage addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double format (see *Data Handling and Precision* on page 168) and placed into FRT.

EA is placed into RA.

Registers Altered

- FRT
- RA

Exceptions

An attempt to execute **lfsux** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

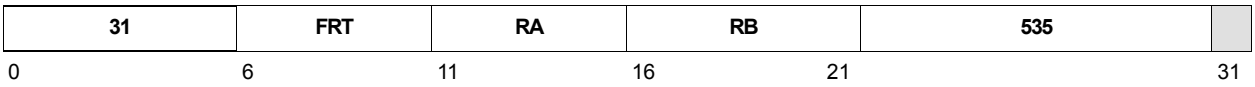
Invalid Forms

RA = 0

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

lfsx FRT, RA, RB



EA ← (RA|0) + (RB)
(FRT) ← DOUBLE(MEM(EA,4))

Let the effective address (EA) be the sum (RA|0) and the index specified by (RB).

The word in storage addressed by EA is interpreted as a floating-point single-precision operand. This word is converted to floating-point double format (see *Data Handling and Precision* on page 168) and placed into FRT.

Registers Altered

- FRT

Exceptions

An attempt to execute **lfsx** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

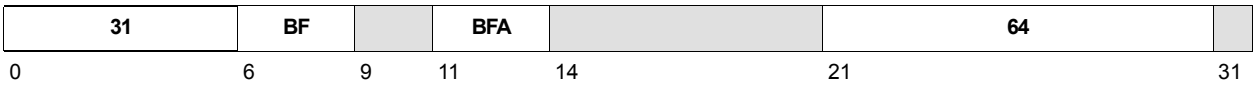
Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

mcrfs

Move to Condition Register from Floating-Point Status and Control

mcrfs BF, BFA



$CR_{BF \times 4:BF \times 4+3} \leftarrow FPSCR_{BFA \times 4:BFA \times 4 + 3}$

$FPSCR_{BFA \times 4:BFA \times 4 + 3} \leftarrow 0b0000$

The contents of the FPSCR specified by BFA are copied to the CR field specified by BF. All exception bits that are copied are set to 0 in the FPSCR. If FPSCR[FX] is copied, it is set to 0 in the FPSCR.

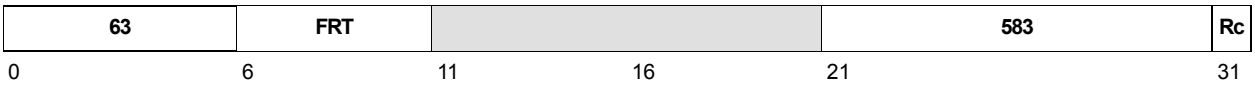
Registers Altered

- CR field BF
- FPSCR[FX, OX] if BFA=0
- FPSCR[UX, ZX, XX, VXSNAN] if BFA=1
- FPSCR[VXISI, VXIDI, VXZDZ, VXIMZ] if BFA=2
- FPSCR[VXVC] if BFA=3
- FPSCR[VXSOFT, VXSQRT, VXCVI] if BFA=5

Exceptions

An attempt to execute **mcrfs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

mffs FRT Rc = 0



(FRT) ← FPSCR

The contents of the FPSCR are placed into $FRT_{32:63}$. $FRT_{0:31}$ are undefined.

Registers Altered

- FRT

Exceptions

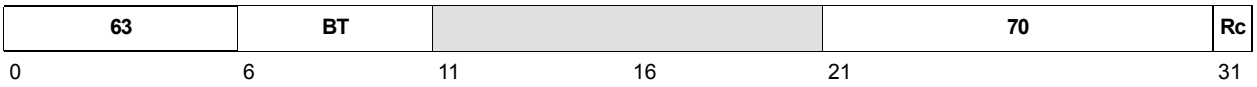
An attempt to execute **mffs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

mtfsb0

Move to Floating-Point Status and Control Register Bit 0

mtfsb0 BT Rc = 0



$FPSCR_{BT} \leftarrow 0$

Bit BT of the FPSCR is set to 0.

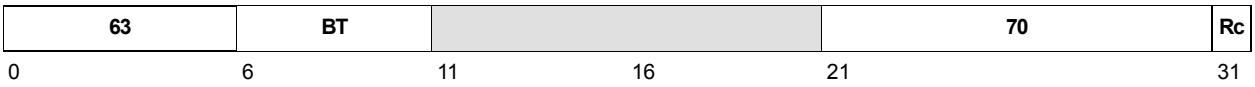
Registers Altered

- FPSCR[BT]

An attempt to execute **mtfsb0** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

mtfsb1 BT Rc = 0



$FPSCR_{BT} \leftarrow 1$

Bit BT of the FPSCR is set to 1.

Registers Altered

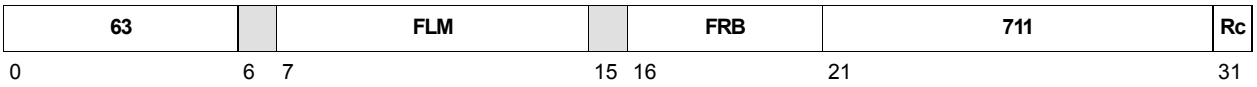
- FPSCR[BT, FX]

AIf Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

Programming Notes

FPSCR[FEX, VX] cannot be explicitly set.

mtfsf FLM, FRB Rc = 0



i ← 0
do while i<8
  if FLM_i=1 then $FPSCR_{4 \times i:4 \times i + 3} \leftarrow (FRB)_{4 \times i:4 \times i + 3}$
  i ← i+1

The contents of bits 32:63 of FRB are placed into the FPSCR under control of the field mask specified by FLM. The field mask identifies the affected 4-bit fields. Let *i* be an integer in the range 0–7. If FLM_i = 1, FPSCR field *i* (FPSCR bits $4 \times i - 4 \times i + 3$) is set to the contents of the corresponding field of the low-order 32 bits of FRB.

FPSCR[FX] is altered only if FLM_0 = 1.

Registers Altered

- FPSCR fields selected by mask

Exceptions

An attempt to execute **mtfsf** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

Programming Notes

Updating fewer than all eight fields of the FPSCR may have substantially poorer performance on some implementations than updating all the fields.

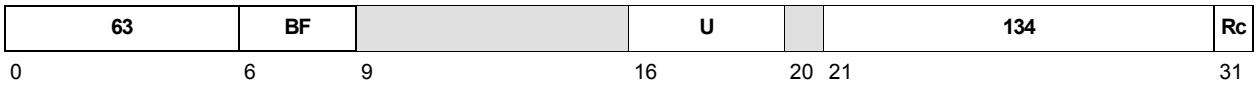
When $FPSCR_{0:3}$ is specified, bits 0 and 3 (FPSCR[FX, OX]) are set to the values of FRB_{32} and FRB_{35}. Even if **mtfsf** causes FPSCR[OX] to change from 0 to 1, FPSCR[FX] is set from FRB_{32}, not by the usual rule that FPSCR[FX] is set to 1 when an exception bit changes from 0 to 1. Bits 1 and 2 (FPSCR[FEX, VX]) are set according to the usual rule, given on XREF TBD, and not from $FRB_{33:34}$.

Implementation Note

In general, the source FPR should contain data produced by a double-precision load operation. The value moved to FPSCR is unpredictable if the source FPR contains denormalized single format data (that is, data that cannot be represented as a normalized single-precision number that was produced from a single precision operation.

This is similar to the achitectural rule for **stfiwx**.

mtfsfi BF, U Rc = 0



$FPSCR_{BF \times 4:BF \times 4 + 3} \leftarrow U$

The value of the U field is placed into FPSCR field BF.

$FPSCR_{FX}$ is altered only if BF = 0.

Registers Altered

- $FPSCR_{BF}$

Exceptions

An attempt to execute **mtfsfi** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

If Rc = 1, an Unimplemented Operation exception type Program interrupt occurs.

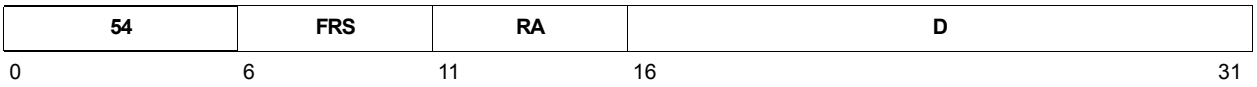
Programming Notes

When $FPSCR_{0:3}$ is specified, bits 0 and 3 FPSCR[FX, OX] are set to the values of U_0 and U_3. Even if **mtfsfi** causes FPSCR[OX] to change from 0 to 1, FPSCR[FX] is set from U_0 and not by the usual rule that FPSCR[FX] is set to 1 when an exception bit changes from 0 to 1). Bits 1 and 2 (FPSCR[FEX, VX]) are set according to the usual rule, given on XREF TBD, and not from $U_{1:2}$.

stfd

Store Floating-Point Double

stfd FRS, D(RA)



EA ← (RA|0) + EXTS(D)
MEM(EA,8) ← (FRS)

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D instruction field to 32 bits.

The contents of FRS are stored into the doubleword in storage addressed by EA.

Registers Altered

- None

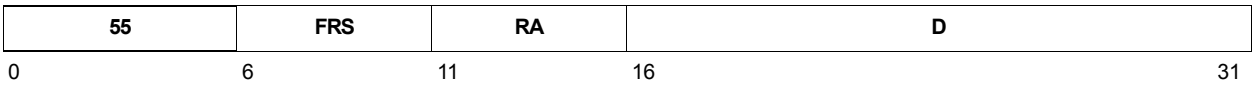
Exceptions

An attempt to execute **stfd** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Implementation Note

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

stfdu FRS, D(RA)



EA ← (RA|0) + EXTS(D)
MEM(EA,8) ← (FRS)
RA ← EA

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D instruction field to 32 bits.

The contents of FRS are stored into the doubleword in storage addressed by EA.

EA is placed into RA.

Registers Altered

- RA

Exceptions

An attempt to execute **stfdu** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

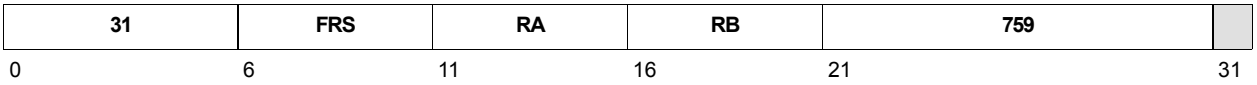
Invalid Forms

RA = 0

Implementation Note

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

stfdux FRS, RA, RB



EA ← (RA|0) + (RB)
MEM(EA,8) ← (FRS)
RA ← EA

Let the effective address (EA) be the sum of (RA|0) and the index specified by (RB).

The contents of FRS are stored into the doubleword in storage addressed by EA.

EA is placed into RA.

Registers Altered

- RA

Exceptions

An attempt to execute **stfdux** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

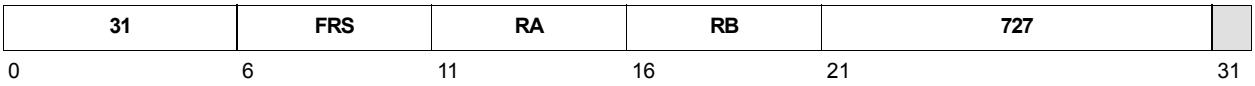
Invalid Forms

RA = 0

Implementation Note

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

stfdx FRS, RA, RB



EA ← (RA|0) + (RB)
MEM(EA,8) ← (FRS)

Let the effective address (EA) be the sum of (RA|0) and the index specified by (RB).

The contents of FRS are stored into the doubleword in storage addressed by EA.

Registers Altered

- None

Exceptions

An attempt to execute **stfdx** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

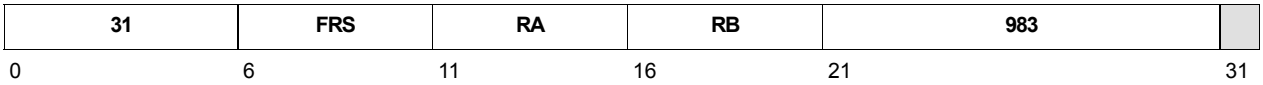
Implementation Note

The effective address (EA) must be 8-byte aligned to prevent Alignment exception.

stfiwx

Store Floating-Point as Integer Word Indexed

stfiwx FRS, RA, RB



EA ← (RA|0) + (RB)
MEM(EA,4) ← $(FRS)_{32:63}$

Let the effective address (EA) be the sum of (RA|0) and the index specified by (RB).

The contents of bits 32:63 of FRS are stored, without conversion, into the word in storage addressed by EA.

If the contents of FRS were produced, either directly or indirectly, by a *Load Floating-Point Single* instruction, a single-precision *Arithmetic* instruction, or **frsp**, the value stored is undefined. (The contents of FRS are produced directly by such an instruction if FRS is the target register for the instruction. The contents of FRS are produced indirectly by such an instruction if FRS is the final target register of a sequence of one or more *Floating-Point Move* instructions, with the input to the sequence having been produced directly by such an instruction.)

Registers Altered

- None

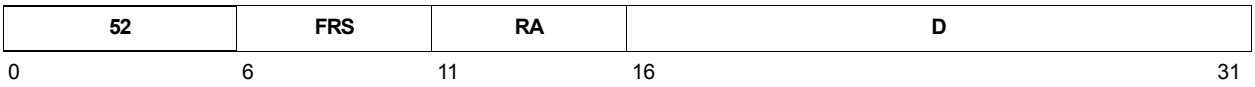
Exceptions

An attempt to execute **stfiwx** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

stfs FRS, D(RA)



EA ← (RA|0) + EXTS(D)
MEM(EA,4) ← SINGLE(FRS)

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D instruction field to 32 bits.

The contents of FRS are converted to single-precision format (see *Data Handling and Precision* on page 168) and stored into the word in storage addressed by EA.

Registers Altered

- None

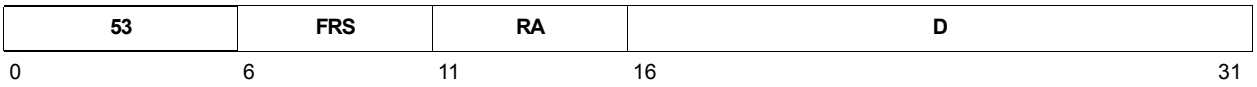
Exceptions

An attempt to execute **stfs** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

stfsu FRS, D(RA)



EA ← (RA|0) + EXTS(D)
MEM(EA,4) ← SINGLE(FRS)
RA ← EA

Let the effective address (EA) be the sum of (RA|0) and the displacement obtained by sign-extending the 16-bit D instruction field to 32 bits.

The contents of FRS are converted to single-precision format (see *Data Handling and Precision* on page 168) and stored into the word in storage addressed by EA.

EA is placed into RA.

Registers Altered

- RA

Exceptions

An attempt to execute **stfsu** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

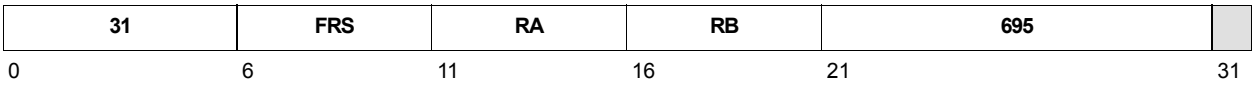
Invalid Forms

RA = 0

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

stfdux FRS, RA, RB



EA ← (RA|0) + (RB)
MEM(EA,4) ← SINGLE(FRS)
RA ← EA

Let the effective address (EA) be the sum of (RA|0) and the index specified by (RB).

The contents of FRS are converted to single-precision format (see *Data Handling and Precision* on page 168) and stored into the word in storage addressed by EA.

EA is placed into RA.

Registers Altered

- RA

Exceptions

An attempt to execute **stfsux** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

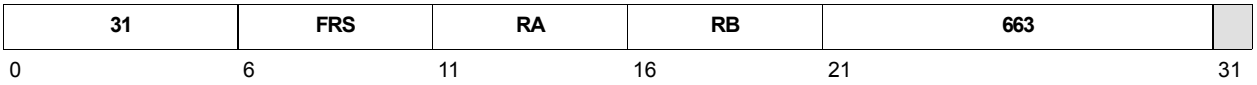
Invalid Forms

RA = 0

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

stfsx FRS, RA, RB



EA ← (RA|0) + (RB)
MEM(EA,4) ← SINGLE(FRS)

Let the effective address (EA) be the sum of (RA|0) and the index specified by (RB).

The contents of FRS are converted to single-precision format (see*Data Handling and Precision* on page 168) and stored into the word in storage addressed by EA.

Registers Altered

- None

Exceptions

An attempt to execute **stfsx** while MSR[FP] = 0 causes a Floating-Point Unavailable exception.

Implementation Note

The EA must be a multiple of 4. Otherwise, an Alignment exception occurs.

42. Register Summary

This chapter summarizes the registers provided by the PPC460EX/EXr/GT that are not associated with the PPC440 processor. These registers include the DCRs, FPRs, MMIO registers, and all registers provided by the peripheral functions.

42.1 Reserved Fields

For all registers with fields marked as reserved, the reserved fields should be written as *zero* and read as *undefined*. That is, when writing to a reserved field, write a zero to that field. When reading from a reserved field, ignore that field.

The recommended coding practice is to perform the initial write to a register with reserved fields as described in the preceding paragraph, and to perform all subsequent writes to the register using a read-modify-write strategy: read the register, alter desired fields with logical instructions, and then write the register.

42.2 Device Control Registers

DCRs may be used to control various on-chip system functions, such as the operation of on-chip buses, peripherals, and certain processor function behaviors. The DCR access instructions are **mtdcr** (move to device control register) and **mfdcr** (move from device control register), which move data between GPRs and the DCRs. Some DCRs are directly accessed, that is, they are accessed using their DCR numbers. Other DCRs are indirectly accessed. Such DCRs are accessed by writing an offset to a directly accessed DCR and then reading the data at the offset in another directly accessed DCR.

42.3 Memory Mapped Registers

Some registers associated with on-chip peripherals are memory-mapped input/output (MMIO) registers. Such registers are mapped into the system memory space and are accessed using load/store instructions that contain the register addresses.

42.4 Alphabetical Summary of Chip Control and Peripheral Function Registers

The following pages provide an alphabetical summary all the device control registers (DCRs) and memory mapped registers (MMIOs) implemented in the PPC460EX/EXr/GT. *Table 42-1* provides the following information:

- Register group mnemonic
- Register type
- Description of the group
- Reference to a page containing details for each register such as the address or DCR number, access, and reset value.

Note: The specific page on which detailed register information is available can be located by finding the register name in the Index. The register name appears twice in the Index–once under "registers" and again under the letter with which the register mnemonic begins.

Table 42-1. Alphabetical Summary of Chip Control and Peripheral Function Register Groups

Register Group	Type	Description	Page
AHB_xxxx	DCR	AHB (PPC460EX only)	140
AHBARB_xxxx	MMIO	AHB Arbiter (PPC460EX only)	148
AHBDMA0_xxxx	MMIO	SATA DMA (PPC460EX only)	717
CPM0_xx	DCR	Clock and Power Management	309
CPR0_CFGxxxx	DCR	Clocking Control	245
CPR0_xxxx	DCR	Clocking Control	246
CRYP0_xxxx	MMIO	Security Function	335
DMA2P40_xxxx	DCR	DMA to PLB4 Controller Configuration and Status	828
EBC0_xxxx	DCR	Exteranal Bus Controller (EBC) Configuration and Status	868
EBC0_xxxx	DCR	EBC DCR Access	868
EHCI0_xxxx	MMIO	Universal Serial Bus 2 (USB2) Host (PPC460EX only)	1292
EMACx_xxxx	MMIO	Ethernet Media AccessController (EMAC) x	1020
FPxxxx	DCR	Floating Point Unit (FPU)	160
GPCSx_xxxx	DCR	Gigabit Mode Physical Coding Sublayer (GPCS)	1040
GPIO0_xxxx	MMIO	General Purpose I/O (GPIO)	1236
GPT0_xxxx	MMIO	General Purpose Time (GPT)	1255
IICn_xxxx	MMIO	Interintegrated Circuit (IIC)	1212
I2O0_xxxx	DCR	Input-to-Output (I2O)/DMA DCRs	908
I2O0_xxxx	MMIO	I2O/DMA	927
I2O0_DMA_xxxx	MMIO	I2O/DMA Memory Mapped DMA	915
L2C0_xxxx	DCR	L2 Cache	192
MAL0_xxxx	DCR	Media Access Layer (MAL) DCRs	973
MCIF0_xxxx	DCR	DDR SDRAM (DDR SDRAM) DCRs	760
NDFC0_xxxx	MMIO	NAND Flash Controller	1279
OHCI0_xxxx	MMIO	Universal Serial Bus 2 (USB2) Host (PPC460EX only)	1292
OPBA0_xxxx	MMIO	On-chip Peripheral Bus (OPB) Arbiter	131
PCI0_xxxx	MMIO	Personal Computer Interface (PCI)	440
PECFGn_xxxx	MMIO	PCI Express Root Port and Endpoint Configuration	587
PEGPLn_xxxx	DCR	PCI Express DCRs	556
PEIH_xxxx	MMIO	PCI Express Interrupt Handler	506
PEUTLn_xxxx	MMIO	PCI Express Application-Specific PLB Address Mapping	566
PLB42OPB0_xxxx	DCR	PLB4 to OPB Bridge	125
PLB4An_xxxx	DCR	PLB4 Arbiter	117
RGMII0_xxxx	MMIO	Reduced GMII (RGMII) Bridge	1063
SATA0_xxxx	MMIO	Serial ATA (PPC460EX only)	663
SDR0_xxxx	DCR	AHB SDRs (PPC460EX only)	136
SDR0_CFGxxxx	DCR	System DCR Configuration	154

Table 42-1. Alphabetical Summary of Chip Control and Peripheral Function Register Groups (Continued)

Register Group	Type	Description	Page
SDR0_ICxxxx	DCR	Interrupt Coalescence	992
SDR0_MALxxx	DCR	MAL SDRs	988
SDR0_PEn_xxxx	DCR	PCIE Configuration DCRs	529
SDR0_PE_TDC	DCR	Target Directed Completion (TDC) DCRs	113
SDR0_xxxx	DCR	IIC Bootstrap Controller	226
SDR0_xxxx	DCR	PLB4 System DCRs	104
SDR0_xxxx	DCR	Security Function DCRs	335
SRAM0_xxxx	DCR	SRAM DCRs	203
SPI0_xxxx	MMIO	Serial Peripheral Interface (SPI)	1507
SRIO0_xxxx	MMIO	Serial RapidIO Interface (SRIO)	1121
TAHx_xxxx	MMIO	TCP/IP Accelerator Hardware	1082
UARTx_xxxx	MMIO	UART Configuration	1094
UICn_xxxx	DCR	UIC Device Control	261
USB0_xxxx	MMIO	USBOTG Global CSRs (PPC460EX only)	1335
USB0_xxxx	MMIO	USBOTG Host Mode CSRs (PPC460EX only)	1335
USB0_xxxx	MMIO	USBOTG Device Mode CSRs (PPC460EX only)	1336
USB0_xxxx	MMIO	USBOTG Clock Gating (PPC460EX only)	1337
USB2HMDCR0_xxxx	DCR	USB AHB-to-OPB (PPC460EX only)	1292
USB2HSDCR0_xxxx	DCR	USB OPB-to-AHB (PPC460EX only)	1292
ZMII0_xxxx	MMIO	ZMII Bridge	1060

43. External Signals

This chapter provides information about the external PPC460EX/EXr/GT signals. All of these signals are described in detail in the *PPC460EX/EXr/GT Embedded Processor Data Sheet*

43.1 Signals Listed Alphabetically

The *Signals Listed Alphabetically* table provides a complete list of all the external PPC460EX/EXr/GT signals in alphabetical order.

43.2 Signals by Interface Category

The *Signal Functional Description* table provides a complete list of all the external PPC460EX/EXr/GT signals grouped together by interface category. The functional characteristics of the signals are described.

43.3 Clocking Signals

Timing information for external clock signals is provided in the *Peripheral Interface I/O Clock Timings* table.

43.4 I/O Signals

I/O timing and output current specifications for the I/O signals are provided in the *I/O Specifications* tables.

Appendix A. Floating Point Instruction Summary

This appendix contains PPC440 FPU instructions summarized alphabetically and by opcode.

Instruction Set – Alphabetical lists all PPC440 FPU mnemonics, including extended mnemonics, alphabetically. A short functional description is included for each mnemonic.

Instructions Sorted by Opcode, lists all PPC440 FPU instructions, sorted by primary and secondary opcodes. Extended mnemonics are not included in the opcode list.

Instruction Formats, illustrates the PPC440 FPU instruction forms (allowed arrangements of fields within instructions).

A.1 Instruction Set – Alphabetical

Table A-1 summarizes the PPC440 FPU instruction set. The instructions are described in detail in *Floating Point Instruction Set* on page 1517 which is also alphabetized under the root form. It also describes the instruction operands and notation.

Table A-1. PPC440 FPU Instruction Syntax Summary

Mnemonic	Operands	Function	Other Registers Changed	Page
fabs	FRT, FRB	Place (FRB), with bit 0 set to zero, into FRT.		1521
fadd	FRT, FRA, FRB	Add (FRA) to (FRB). Place result in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI]	1522
fadds	FRT, FRA, FRB	Add (FRA) to (FRB). Round result to single-precision format and place in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI]	1523
fcmpo	BF, FRA, FRB	(FRA) is compared to (FRB). Place result in CR[BF] and FPSCR[FPCC].	CR[BF FPSCR[FPCC, FX, VXSNAN, VXVC]	1524
fcmpu	BF, FRA, FRB	(FRA) is compared to (FRB). Place result in CR[BF] and FPSCR[FPCC].	CR[BF] FPSCR[FPCC, FX, VXSNAN]	1525
fctiw	FRT, FRB	Convert (FRB) to a 32-bit signed integer. Place result in $FRT_{32:63}$. $FRT_{0:31}$ are undefined.	FPSCR[FR, FI, FX, XX, VXSNAN, VXCVI] FPSCR[FPRF] undefined	1526
fctiwz	FRT, FRB	Convert (FRB) to a 32-bit signed integer. Place result in $FRT_{32:63}$. $FRT_{0:31}$ are undefined.	FPSCR[FR, FI, FX, XX, VXSNAN, VXCVI] FPSCR[FPRF] undefined	1527
fdiv	FRT, FRA, FRB	Divide (FRA) by (FRB). Place result in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ]	1528
fdivs	FRT, FRA, FRB	Divide (FRA) by (FRB). Round result to single-precision format and place in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, ZX, XX, VXSNAN, VXIDI, VXZDZ]	1529
fmadd	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Add (FRB) to the intermediate result. Place result in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]	1530

Table A-1. PPC440 FPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
fmadds	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Add (FRB) to the intermediate result. Round result to single-precision format and place in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]	1531
fmsub	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Subtract (FRB) from the intermediate result. Place result in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI]	1533
fmsubs	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Subtract (FRB) from the intermediate result. Round result to single-precision format and place in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI]	1534
fmul	FRT, FRA, FRC	Multiply (FRA) by (FRC). Place result in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXIMZ]	1535
fmuls	FRT, FRA, FRC	Multiply (FRA) by (FRC). Round result to single-precision format and place in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXIMZ]	1536
fnabs	FRT, FRB	Place (FRB), with bit 0 set to one, into FRT.		1537
fneg	FRT, FRB	Place (FRB), with bit 0 inverted, into FRT.		1538
fnmadd	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Add (FRB) to the intermediate result. Negate result and place it in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]	1539
fnmadds	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Add (FRB) to the intermediate result. Negate result, round it to single-precision format and place in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]	1531
fnmsub	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Subtract (FRB) from the intermediate result. Negate result and place it in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]	1541
fnmsubs	FRT, FRA, FRC, FRB	Multiply (FRA) by (FRC). Subtract (FRB) from the intermediate result. Negate result, round it to single-precision format and place in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN, VXISI, VXIMZ]	1542
fres	FRT, FRB	A single-precision estimate of the reciprocal of the floating-point operand in FRB is placed into FRT.	FPSCR[FX, OX, UX, ZX, VXSNAN], FPSCR[FPRF, FR, FI] undefined	1543
frsp	FRT, FRB	Round (FRB) to single precision and place the result in FRT.	FPSCR[FPRF, FR, FI, FX, OX, UX, XX, VXSNAN]	1544
frsqrte	FRT, FRB	A double-precision estimate of the reciprocal of the square root of the floating-point operand in FRB is placed into FRT.	FPSCR[FX, ZX, VXSNAN, VXSQRT], FPSCR[FPRF, FR, FI] undefined	1545
fsel	FRT, FRA, FRC, FRB	The floating-point operand in FRA is compared to zero.		1547
fsub	FRT, FRA, FRB	The floating-point operand in FPR(FRB) is subtracted from the floating-point operand in FPR(FRA).		1548
fsubs	FRT, FRA, FRB	The floating-point operand in FPR(FRB) is subtracted from the floating-point operand in FPR(FRA).	FPSCR[FPRF, FR, FI FX, OX, UX, XX, VXSNAN]	1549
lfd	FRT, D(RA)	Load the double word from EA = (RA|0) + EXTS(D) into FRT.		1550
lfdu	FRT, D(RA)	Load the double word from EA = (RA|0) + EXTS(D) into FRT. Update the base address, (RA) ← EA		1551

Table A-1. PPC440 FPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
lfdux	FRT, RA, RB	Load the double word from EA = (RA|0) + (RB) into FRT. Update the base address, (RA) ← EA.		1552
lfdx	FRT, RA, RB	Load the double word from EA = (RA|0) + (RB) into FRT.		1553
lfs	FRT, D(RA)	Load the word from EA = (RA|0) + EXTS(D) into FRT. The word is interpreted as a single precision operand.		1554
lfsu	FRT, D(RA)	Load the word from EA = (RA|0) + EXTS(D) into FRT. The word is interpreted as a single precision operand. Update the base address, (RA) ← EA.		1555
lfsux	FRT, RA, RB	Load the word from EA = (RA|0) + (RB) into FRT. The word is interpreted as a single precision operand. Update the base address, (RA) ← EA.		1556
lfsx	FRT, RA, RB	Load the word from EA = (RA|0) + (RB). The word is interpreted as a single precision operand.		1557
mcrfs	BF, BFA	Copy the contents of the FPSCR specified by BFA to the CR field specified by BF. Place result in CR[BF] and FPSCR[FPCC].	CR[BF See the instruction description for details of changed FPSCR fields	1558
mffs	FRT	Place (FPSCR) into $FRT_{32:63}$. $FRT_{0:31}$ are undefined.		1559
mtfsb0	BT	Set the FPSCR bit specified by BT to 0.	$FPSCR_{BT}$	1560
mtfsb1	BT	Set the FPSCR bit specified by BT to 1.	$FPSCR_{BT}$	1561
mtfsf	FLM, FRB	Place $FRB_{32:63}$ in the FPSCR under control of the mask specified by FLM.	$FPSCR_{BF}$	1562
mtfsfi	FLM, FRB	Place $FRB_{32:63}$ in the FPSCR under control of the mask specified by FLM.	FPSCR fields altered by mask.	1563
stfd	FRS, D(RA)	Store double word (FRS) at EA = (RA|0) + EXTS(D).		1564
stfdu	FRS, D(RA)	Store double word (FRS) at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		1565
stfdux	FRS, RA, RB	Store double word (FRS) at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		1566
stfdx	FRS, RA, RB	Store double word (FRS) at EA = (RA|0) + (RB).		1567
stfiwx	FRS, RA, RB	Store word $FRS_{32:63}$ at EA = (RA|0) + (RB).		1568
stfs	FRS, D(RA)	Convert (FRS) to single-precision format and store at EA = (RA|0) + EXTS(D).		1569
stfsu	FRS, D(RA)	Convert (FRS) to single-precision format and store at EA = (RA|0) + EXTS(D). Update the base address, (RA) ← EA.		1570

Table A-1. PPC440 FPU Instruction Syntax Summary (Continued)

Mnemonic	Operands	Function	Other Registers Changed	Page
stfsux	FRS, RA, RB	Convert (FRS) to single-precision format and store at EA = (RA|0) + (RB). Update the base address, (RA) ← EA.		1571
stfsx	RS, RA, RB	Convert (FRS) to single-precision format and store at EA = (RA|0) + (RB).		1572

A.2 Instructions Sorted by Opcode

All instructions are four bytes long and word aligned. All instructions have a primary opcode field (shown as field OPCD in *Figure A-1* through *Figure A-10*, beginning on page 1587) in bits 0:5. Some instructions also have a secondary opcode field (shown as field XO in *Figure A-1* through *Figure A-10*). PPC440 FPU instructions, sorted by primary and secondary opcode, are listed in *Table A-2*.

The "Form" indicated in the table refers to the arrangement of valid field combinations within the four-byte instruction. See *Instruction Formats* on page 1584, for the field layouts of each form.

Form X has a 10-bit secondary opcode field, while form XO uses only the low-order 9-bits of that field. Form XO uses the high-order secondary opcode bit (the tenth bit) as a variable; therefore, every form XO instruction really consumes two secondary opcodes from the 10-bit secondary-opcode space. The implicitly consumed secondary opcode is listed in parentheses for form XO instructions in the table below.

Table A-2. PPC440 FPU Instructions by Opcode

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
31	535	X	**lfsx**	FRT, RA, RB	1557
31	567	X	**lfsux**	FRT, RA, RB	1556
31	599	X	**lfdx**	FRT, RA, RB	1553
31	631	X	**lfdux**	FRT, RA, RB	1552
31	695	X	**stfsux**	FRS, RA, RB	1571
31	727	X	**stfdx**	FRS, RA, RB	1567
31	759	X	**stfdux**	FRS, RA, RB	1566
31	983	X	**stfiwx**	FRS, RA, RB	1568
48		D	**lfs**	FRT, D(RA)	1554
49		D	**lfsu**	FRT, D(RA)	1555
50		D	**lfd**	FRT, D(RA)	1550
51		D	**lfdu**	FRT, D(RA)	1551
52		D	**stfs**	FRT, D(RA)	1569
53		D	**stfsu**	FRT, D(RA)	1570
54		D	**stfd**	FRS, D(RA)	1564
55		D	**stfdu**	FRS, D(RA)	1565
59	18	A	fdivs	FRT, FRA, FRB	1529

Table A-2. PPC440 FPU Instructions by Opcode (Continued)

Primary Opcode	Secondary Opcode	Form	Mnemonic	Operands	Page
59	20	A	fsubs	FRT, FRA, FRB	1549
59	21	A	fadds	FRT, FRA, FRB	1523
59	25	A	fmuls	FRT, FRA, FRC	1536
59	28	A	fmsubs	FRT, FRA, FRC, FRB	1534
59	29	A	fmadds	FRT, FRA, FRC, FRB	1540
59	30	A	fnmsubs	FRT, FRA, FRC, FRB	1542
59	31	A	fnmadds	FRT, FRA, FRC, FRB	1540
63	12	X	frsp	FRT, FRB	1544
63	14	X	fctiw	FRT, FRB	1526
63	15	X	fctiwz	FRT, FRB	1527
63	18	A	fdiv	FRT, FRA, FRB	1528
63	20	A	fsub	FRT, FRA, FRB	1548
63	21	A	fadd	FRT, FRA, FRB	1522
63	25	A	fmul	FRT, FRA, FRC	1535
63	28	A	fmsub	FRT, FRA, FRC, FRB	1533
63	29	A	fmadd	FRT, FRA, FRC, FRB	1530
63	30	A	fnmsub	FRT, FRA, FRC, FRB	1541
63	31	A	fnmadds	FRT, FRA, FRC, FRB	1540
63	38	X	mtfsb1	BT	1561
63	40	X	**fneg**	FRT, FRB	1538
63	70	X	mtfsb0	BT	1560
63	72	X	fmr	FRT, FRB	1532
63	134	X	mtfsfi	BF, U	1563
63	136	X	fnabs	FRT, FRB	1537
63	264	X	fabs	FRT, FRB	1521
63	583	X	mffs	FRT	1559
63	711	XFL	mtfsf	BF, U	1562

A.3 Instruction Formats

Instructions are four bytes long. Instruction addresses are always word-aligned.

Instruction bits 0 through 5 always contain the primary opcode. Many instructions have an extended opcode in another field. Remaining instruction bits contain additional fields. All instruction fields belong to one of the following categories:

- Defined

  These instructions contain values, such as opcodes, that cannot be altered. The instruction format diagrams specify the values of defined fields.

- Variable

  These fields contain operands, such as GPR selectors and immediate values, that can vary from execution to execution. The instruction format diagrams specify the operands in the variable fields.

- Reserved

  Bits in reserved fields should be set to 0. In the instruction format diagrams, /, //, or /// indicate reserved fields.

If any bit in a defined field does not contain the expected value, the instruction is illegal and an illegal instruction exception occurs. If any bit in a reserved field does not contain 0, the instruction form is invalid; its result is architecturally undefined. The PPC440 FPU executes all invalid instruction forms without causing an illegal instruction exception.

A.3.1 Instruction Fields

PPC440 FPU instructions contain various combinations of the following fields, as indicated in the instruction format diagrams that follow the field definitions. Numbers, enclosed in parentheses, that follow the field names indicate bit positions; bit fields are indicated by starting and stopping bit positions separated by colons.

AA (30) Absolute address bit.

- 0 The immediate field represents an address relative to the current instruction address (CIA). The effective address (EA) of the branch is either the sum of the LI field sign-extended to 32 bits and the branch instruction address, or the sum of the BD field sign-extended to 32 bits and the branch instruction address.
- 1 The immediate field represents an absolute address. The EA of the branch is either the LI field or the BD field, sign-extended to 32 bits.

BA (11:15) Specifies a bit in the CR used as a source of a CR-logical instruction.

BB (16:20) Specifies a bit in the CR used as a source of a CR-logical instruction.

BD (16:29) An immediate field specifying a 14-bit signed twos complement branch displacement. This field is concatenated on the right with 0b00 and sign-extended to 32 bits.

BF (6:8) Specifies a field in the CR used as a target in a compare or **mcrf** instruction.

BFA (11:13) Specifies a field in the CR used as a source in a **mcrf** instruction.

BI (11:15) Specifies a bit in the CR used as a source for the condition of a conditional branch instruction.

BO (6:10) Specifies options for conditional branch instructions.

BT (6:10) Specifies a bit in the CR used as a target as the result of a CR-Logical instruction.

D (16:31) Specifies a 16-bit signed twos-complement integer displacement for load/store instructions.

DCRN (11:20) Specifies a device control register (DCR).

FXM (12:19) Field mask used to identify CR fields to be updated by the **mtcrf** instruction.

IM (16:31) An immediate field used to specify a 16-bit value (either signed integer or unsigned).

LI (6:29) An immediate field specifying a 24-bit signed twos complement branch displacement; this field is concatenated on the right with b'00' and sign-extended to 32 bits.

LK (31) Link bit.

0 Do not update the link register (LR).

1 Update the LR with the address of the next instruction.

MB (21:25) Mask begin.

Used in rotate-and-mask instructions to specify the beginning bit of a mask.

ME (26:30) Mask end.

Used in rotate-and-mask instructions to specify the ending bit of a mask.

NB (16:20) Specifies the number of bytes to move in an immediate string load or store.

OPCD (0:5) Primary opcode. Primary opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The OPCD field name does not appear in instruction descriptions.

OE (21) Enables setting the OV and SO fields in the fixed-point exception register (XER) for extended arithmetic.

RA (11:15) A GPR used as a source or target.

RB (16:20) A GPR used as a source.

Rc (31) Record bit.

0 Do not set the CR.

1 Set the CR to reflect the result of an operation.

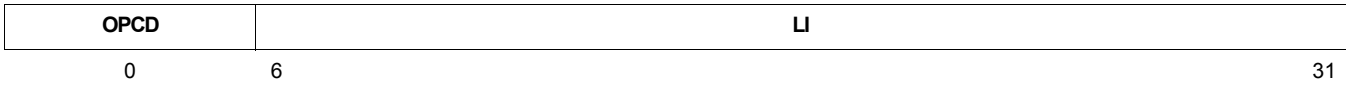
RS (6:10)	A GPR used as a source.
RT (6:10)	A GPR used as a target.
SH (16:20)	Specifies a shift amount.
SPRF (11:20)	Specifies a special purpose register (SPR).
TO (6:10)	Specifies the conditions on which to trap, as described under **tw** and **twi** instructions.
XO (21:30)	Extended opcode for instructions without an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.
XO (22:30)	Extended opcode for instructions with an OE field. Extended opcodes, in decimal, appear in the instruction format diagrams presented with individual instructions. The XO field name does not appear in instruction descriptions.

A.3.2 Instruction Format Diagrams

The instruction formats (also called "forms") illustrated in *Figure A-1* through *Figure A-10* are valid combinations of instruction fields. *Table A-2* on page 1582 indicates which "form" is utilized by each PPC440 FPU opcode. Fields indicated by slashes (/, //, or ///) are reserved. The figures are adapted from the PowerPC User Instruction Set Architecture.

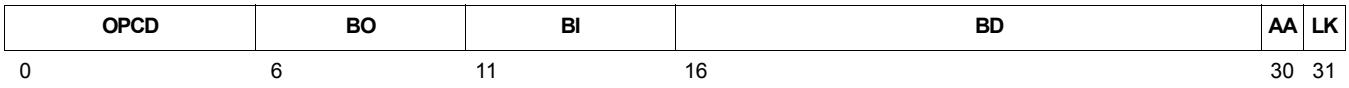
A.3.2.1 I-Form

Figure A-1. I Instruction Format



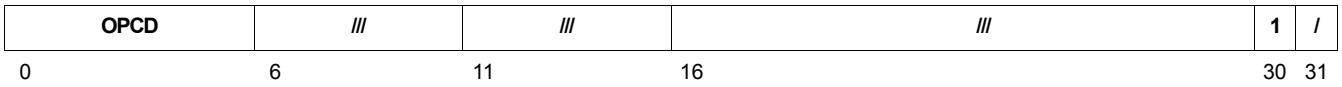
A.3.2.2 B-Form

Figure A-2. B Instruction Format



A.3.2.3 SC-Form

Figure A-3. SC Instruction Format



A.3.2.4 D-Form

Figure A-4. D Instruction Format

OPCD	RT		RA	D
OPCD	RS		RA	SI
OPCD	RS		RA	D
OPCD	RS		RA	UI
OPCD	BF	/ L	RA	SI
OPCD	BF	/ L	RA	UI
OPCD	TO		RA	SI

0 6 11 16 31

A.3.2.5 X-Form

Figure A-5. X Instruction Format

OPCD	RT	RA	RB	XO	Rc
OPCD	RT	RA	RB	XO	/
OPCD	RT	RA	NB	XO	/
OPCD	RT	RA	WS	XO	/
OPCD	RT	///	RB	XO	/
OPCD	RT	///	///	XO	/
OPCD	RS	RA	RB	XO	Rc
OPCD	RS	RA	RB	XO	1
OPCD	RS	RA	RB	XO	/
OPCD	RS	RA	NB	XO	/
OPCD	RS	RA	WS	XO	/
OPCD	RS	RA	SH	XO	Rc
OPCD	RS	RA	///	XO	Rc
OPCD	RS	///	RB	XO	/
OPCD	RS	///	///	XO	/
OPCD	BF / L	RA	RB	XO	/
OPCD	BF //	BFA //	///	XO	Rc
OPCD	BF //	///	///	XO	/
OPCD	BF //	///	U	XO	Rc
OPCD	BF //	///	///	XO	/
OPCD	TO	RA	RB	XO	/
OPCD	BT	///	///	XO	Rc
OPCD	///	RA	RB	XO	/
OPCD	///	///	///	XO	/
OPCD	///	///	E //	XO	/
0	6	11	16	21	31

A.3.2.6 XL-Form

Figure A-6. XL Instruction Format

OPCD	BT	BA	BB	XO	/
OPCD	BC	BI	///	XO	LK
OPCD	BF //	BFA //	///	XO	/
OPCD	///	///	///	XO	/
0	6	11	16	21	31

A.3.2.7 XFL-Form

Figure A-7. XFX Instruction Format

OPCD	/	FLM	/	FRB	XO	Rc
0	6	15	16	21		31

A.3.2.8 XFX-Form

Figure A-8. XFX Instruction Format

OPCD	RT	SPRF	XO	/	
OPCD	RT	DCRF	XO	/	
OPCD	RT	/ FXM /	XO	/	
OPCD	RS	SPRF	XO	/	
OPCD	RS	DCRF	XO	/	
0	6	11 16	21		31

A.3.2.9 X0-Form

Figure A-9. XO Instruction Format

OPCD	RT	RA	RB	OE	XO	Rc
OPCD	RT	RA	RB	OE	XO	Rc
OPCD	RT	RA	///	/	XO	Rc
0	6	11	16	21	22	31

A.3.2.10 M-Form

Figure A-10. M Instruction Format

OPCD	RS	RA	RB	MB	ME	Rc
OPCD	RS	RA	SH	MB	ME	Rc
0	6	11	16	21	26	31

Index

A

B

C

D

E

F

G

H, I, J, K

L

M

N

O

P

Q

R

S

T

U, V, W

Z

Revision Log

Revision Level	Date	Contents of Modification
1.00	Oct. 02, 2007	Initial creation.
1.01	Oct. 31, 2007	Updates to initial version.
1.02	Nov. 12, 2007	Updates to initial version.
1.03	Dec. 17, 2007	Updates to initial version including SATA without SATA DMA.
1.04	Jan. 11, 2008	Updates to initial version including SATA with SATA DMA.
1.05	Jan. 25, 2008	Updates to initial version. Updates to DDR SDRAM section Add three missing Ethernet SDR0 registers to Section 14. Add second USB section for OTG. Delete AHB as a separate section and include in on-chip bus section.
1.06	Feb. 25, 2008	Clean up for consistent formatting. Doc Issue 465. Correct interrupt specs. Doc Issue 473. Replace DMA section (Doc Issue 475). Updates to GPIO section (Doc Issue 480).
1.07	Mar. 03, 2008	Updates to SDRAM section (Doc Issue 485). Update to EMAC section (Doc Issue 487). Convert SATA DMA registers from 64 to 32 bits.
1.08	Mar. 14, 2008	Add Interrupt Coalescing to MAL. Misc. updates including USB and elimination of leftover references to the 405 processor.
1.09	Apr. 1, 2008	Update to *Processor State After Reset* on page 286. Replace the "core" with the "function" where appropriate. Misc. updates to Sections 7 (L2 Cache) and 8 (SRAM Controller). Convert the UM from 460EX-only to a combined 460EX and 460GT UM.
1.10	Apr. 9, 2008	Eliminate PCC405 material and references. Substitute references to the DS for duplicate material originally from the DSs. Updates to correctly mark 460EX only items.
1.11	May 5, 2008	Restore USB DMA and thresholding. Change EMAC-to-PHY diagram. Doc Issue 528
1.12	June 4, 2008	Add SRIO. Misc. updates.
1.13	June 30, 2008	Misc. updates. Doc Issue 546. Update EMAC section.

Continued on next page.

Revision Level	Date	Contents of Modification
1.14	Sept. 2, 2008	Misc. updates (spelling, formatting, punctuation, etc.). Remove 460GT warning from title page. Change document classification from Advanced to Preliminary. Doc Issue 547. Updates to L2 Cache section. Change BxAP in Nand Flash chapter to BnAp to match EBC register name. Doc Issue 557. Update MAL0_ESR[CIDT]. Doc Issue 560. Update EMACx_MR1[JPSM]. Doc Issue 561. Update EMACx_RMR[ROP]. Doc Issue 562-1. Update EMACx_TMR1[TLR]. Doc Issue 562-2. Update section 28.10.30.1. Doc Issue 565. Update GPIOx_OSRL and GPIOx_OSRH. Misc. updates from Engineering. Doc Issue 568. Update NAND Flash section. Doc Issue 571. Updates to DDR SDRAM section. Doc Issue 579. Update EMACx_TMR1 and EMACx_RMR. SRIO updates from engineering. More misc. updates from engineering. Remove "optional" from Security. Doc Issue 587. Update Hash register bit definitions. Add new RAID5 section. Additional corrections to DDR SDRAM section. Change "Rapid I/O" to "RapidIO" and other misc. changes requested by engineering.
1.15	Nov. 14, 2008	Doc Issue 598. Move KASUMI algorithm from its own section into the Security section. Update **ifd** and **stfd** registers in the Floating Point register summary. Doc Issue 603. Update PCIE DCR base address. Doc Issue 602. Add USB0_DTHRCTL register to USB chapter. Bugzilla Issue 4868. Correct bit definitions in registers USB0_DAINT and USB0_DAINTMSK. Bugzilla Issue 4958. Remove External Bus Master references in EBC section. Bugzilla Issue 5037. Correct value written to reset ECC errors. Bugzilla Issue 5044. Update SRAMx_SBnCR. Bugzilla Issue 5039. Updates to Chapter 30 (ZMII/RGMII bridges). Bugzilla Issue 5098. Change 0 and 1 assignments in GPIOx_TCR register. Bugzilla Issue 5190. Correct EMAC interrupt UIC bit numbers. Bugzilla Issue 5204. Corrrect SysReset defintiion.
1.16	Nov. 17, 2008	Update to include 460EXr. Change processor references to 440H6.
1.17	Feb. 13, 2009	Bugzilla Issue 5418. Added *I2O MSI Mapping Register (PEIH_MSIMAP)* on page 508. Bugzilla Issue 5460. Changed 'PLVEDV0' to 'PLBEDV0' in *Figure 10-1* on page 239. Updated description in *Clocking Update Register (CPR0_CLKUPD)* on page 247. Bugzilla Issue 5573. Changed 'CCR0[TCS]' to 'CCR1[TCS]' in *Source Clocks for UART, Timers, MDIO, IIC and SPI* on page 242. Bugzilla Issue 5593. Added reset values and clarified some bit settings in *SPI Registers* on page 1507. Bugzilla Issue 5597. Corrected DCR address for PCIe port 0 and part 1 in *Figure 20-30* on page 528. Changed DCR addresses from 0x40 and 0x60 to 0x100 and 0x120, respectively. Bugzilla Issue 5750. Changed 'EBC0_CFC[OEO] - 0' to 'EBC0_CFG[OEO] = 0' in *Peripheral Bank Access Parameters (EBC0_B0AP–EBC0_B5AP)* on page 870. Bugzilla Issue 5770. Swapped bit settings in *GPIO Open Drain Register (GPIOx_ODR)* on page 1239. Bugzilla Issue 5875. Changed 'TLR > TUR > TFS ≥ 17 entries' to 'TFS > TUR > TLR ≥ 17 entries' in *Transmit Mode Register 1 (EMACx_TMR1)* on page 1025.

Continued on next page.

Revision Level	Date	Contents of Modification
1.18	June 25, 2009	Doc Issue 5900: Updated Chapter 10, "Clocking" (PerClk feedback not recommended). Doc Issue 5909: Updated Chapter 28, "Ethernet Media Access Controller" with information on QoS (virtual MAL RX channels). Doc Issue 5911: Update to EBC error handling in Chapter 25, "External Bus Controller". Doc Issue 5924: Change B to bytes in Chapter 28, "Ethernet Media Access Controller". Doc Issue 5927: Updated name of register EHCI0_INSNREG5 in *Figure 37-14* on page 1310 Doc Issue 5930: Added new section called *UART System Device Control Registers* on page 1102 in Chapter 31, "UART Serial Port Operations". Doc Issue 5935 and Doc Issue 6660: Updates to Chapter 9, "Bootstrap Operations". Doc Issue 5937: Added UART divisors for 200 MHz to *Table 31-1* on page 1091. Doc Issue 5985: Replaced all references to SDR0_PEn_RCSSET[RSTPYN] in Chapter 20, "PCI Express (PCIE)". Doc Issue 5987: Fixed *Table 34-7* on page 1243 in the Chapter 34, "GPIO Operations" (NAND flash signals are not alternate 1s). Doc Issue 6362: Fixed typo on GCM in Chapter 18, "Security Function". Doc Issue 6453: Updated Chapter 36, "NAND Flash Controller" regarding NAND Flash sizes supported and boot from NAND Flash. Doc Issue 6611: Updated addresses for AHB Arbiter Registers (*Table 2-17* on page 148). Doc Issue 6634: • Updated EMACx_MR1[OBCI] bit field description in *Mode Register 1 (EMACx_MR1)* on page 1022. • Corrections to *Table 10-1* on page 238 and *Table 10-7* on page 244 in the Chapter 10, "Clocking". • Defined GBIF (Generic Bus Interface) abbreviation in *Figure 20-1* on page 482. • Corrected bit field description of GPIOx_TCR in *Figure 34-3* on page 1237. • Corrected SRAMx_DPC reset value and added recommendation to initialize SRAM parity bits to *SRAM Data Parity Checking Register (SRAMx_DPC)* on page 210. • In Chapter 19, "PCI Controller (PPC460EX-N and PPC460GT-N Only)", updated the description of the PCI0_CFGADDR and PCI0_CFGDATA registers and their use for configuration head access. • Updated the description of the *PCI Arbiter* on page 431. Corrected the number of REQ/GNT signals. • Made updates to *Figure 20-30* on page 528. • Chapter 22, "Double Data Rate (DDR) SDRAM Controller" updates: • Updated introduction. • Replaced register prefixes SDRAM0_ with MCIF0_. • Replaced bank with rank where appropriate. • Added tables listing commonly supported DDR memories. • Added tables to describe the example MCIF0_MODTn configuration for DIMMs. Doc Issue 6646: Removed iSCSI CRC32 support from Chapter 26, "I2O/DMA (HSDMA) Controller". Doc Issue 6647: Removed SGMII support on 460EXr. Moved the *PPC460EX/EXr RAID Addendum* to Chapter 23, "RAID5 (PPC460EX/EXr Only)".
Continued on next page.		

Revision Level	Date	Contents of Modification
1.19	November 04, 2009	Removed Preliminary classification. Doc Issue 6098: Changed definition for CDR_FREQLOOP_GAIN values in SDR0_PE0_L0CDRCTL (see *Figure 20-41* on page 540 and *Figure 20-42* on page 541). Doc Issues 6245/6694: Updated the PLB addresses for the PCI MST registers in *Table 19-6* on page 443. Doc Issue 6622: Complete update of Chapter 27, "Memory Access Layer". Doc Issue 6713: Added additional detail on Public Key Acceleration (PKA) engine to *Public Key Accelerator and Public Key Memory* on page 322. Doc Issue 6740: Changes to MCIF0_CODT (DQLZ and CKLZ bitfields) in *Figure 22-22* on page 772 and to Notes under *Table 22-14* on page 774. Doc Issue 6783: Put the correct address for register CRYP0_BYTE_ORDER in *Table 18-13* on page 335. Doc Issue 6788: Corrected reserved bit field range in *Figure 9-12* on page 234. Bits 4:16 of SDR0_EBC0 are reserved. Doc Issue 6871: Swap 0 and 1 bit definitions for MAL Rx Descriptor "I" bit field in *Table 27-3* on page 963. Doc Issue 6873: Corrected USB EHCI and OHCI memory mapped register addresses in *Table 37-2* on page 1292. Added comments indicating USB OTG base address to *CSR Memory Map* on page 1334. Doc Issue 6877: Updates on L2 Cache snooping functionality in Chapter 7, "L2 Cache/SRAM Controller". Doc Issue 6935: Add support for port SGMII1 on 460EXr. Added new bit fields to *Ethernet Configuration Register (SDR0_ETH_CFG)* on page 293 register for SGMII auto-negotiate. New bit fields enable auto-negotiate on 460EX/EXr/GT rev B. Replaced all references to XAUI with SRIO in Chapter 20, "PCI Express (PCIE)". Doc Issue 6938: Added SPI timing diagrams to Chapter 39, "Serial Peripheral Interface (SPI)". Doc Issue 6789/7097: Corrected the bit field description of CPR0_PLB2D[PLB2DV0] in *Figure 10-9* on page 250. CPR0_PLB2D[PLB2DV0] must be set to 1. Added notes about the DBCR0[RST] in Chapter 14, "Reset and Initialization". DBCR0[RST] can be used to generate System, Chip, or CPU "Core" reset. Added a note to *Figure 14-1* on page 285 recommending the use of SysReset instead of ExtReset for boot memory. Doc Issue 7126: Corrections for *Built-In Self Test (BIST) for PCI Express Ports* on page 524 (register names, bitfield names, and instructions/steps). Doc Issue 7141: Corrected Ethernet Transmit FIFO sizes for ports EMAC0 and EMAC1.
1.20	November 10, 2009	Doc Issue 7505: Additional corrections for *Built-In Self Test (BIST) for PCI Express Ports* on page 524 (register names, bitfield names, and instructions/steps).
1.21	December 9, 2009	Corrected document error introduced in revision 1.18 (erroneous references to "PPC405 FPU" changed back to "PPC440 FPU").

Revision Level	Date	Contents of Modification
1.22	March 30, 2012	Doc Issue 5874: Updated *Section 25.5.3 , Peripheral Bank Configuration Registers (EBC0_B0CR–EBC0_B5CR), on page 869*. Doc Issue 7386: Updated *Section 20.21.14 , Outbound Read Tag Allocation Register (PEUTLn_OUTTR), on page 574* and *Section 20.21.15 , Inbound Read Tag Allocation Register (PEUTLn_INTR), on page 575*. Doc Issue 7812: Updated *Section 19.5.1.2 , PCI0_CFGADDR Register Field Description, on page 414*. Doc Issue 7900: Updated *Section 18.1.2 , PLB Interface, on page 317* and *Section 18.4.19 , PE Ring Poll Register (CRYP0_PE_RING_P), on page 359*. Doc Issue 8088: Updated *Section 19.17 , PCI–PLB Relative Clock Frequency Requirements, on page 480*, *Section 28.3.8 , EMAC Loopback Modes, on page 1001*, and *Section 28.10.1 , Mode Register 0 (EMACx_MR0), on page 1021*. Doc Issue 9223: Updated *Section 25.4.4 , Device-Paced Burst Write Transfer, on page 866*. Doc Issue 9226: Updated *Section 34.4.3 , GPIO Three-State Control Register (GPIOx_TCR), on page 1237*. Doc Issue 9709: Updated *Section 28.10.3 , Transmit Mode Register 0 (EMACx_TMR0), on page 1024*, and *Section 28.10.4.2 , Urgent-Priority Requests, on page 1025*. Doc Issue 9825: Updated *Section 19.14.4.4 , PCI Status Register (PCI0_STATUS), on page 445*. Doc Issue 10099: Added new sections to Chapter 33. *Section 33.1 , Overview of the I2C Bus, on page 1189*. Updated section *Section 33.9.5 , IICx Control Register (IICx_CNTL), on page 1216*. Doc Issue 10166: Updated *Table 22-3*, *Table 22-4*, and *Table 22-5*. Updated sections *22.2.2*, *22.2.8.1*, *22.2.8.3*, *22.2.8.4*, *22.2.8.7*, *22.2.8.9*, *22.2.8.10*, *22.2.8.12*, and *22.2.9*. Doc Issue 10395: Updated *Table 42-1, Alphabetical Summary of Chip Control and Peripheral Function Register Groups*. Doc Issue 10440: Updated *Figure 27-13, Receive Channel Active Reset Register (MAL0_RXCARR)*. Doc Issue 10676: Updated *Figure 38-5, AHB Configuration Register (USB0_GAHBCFG)*.