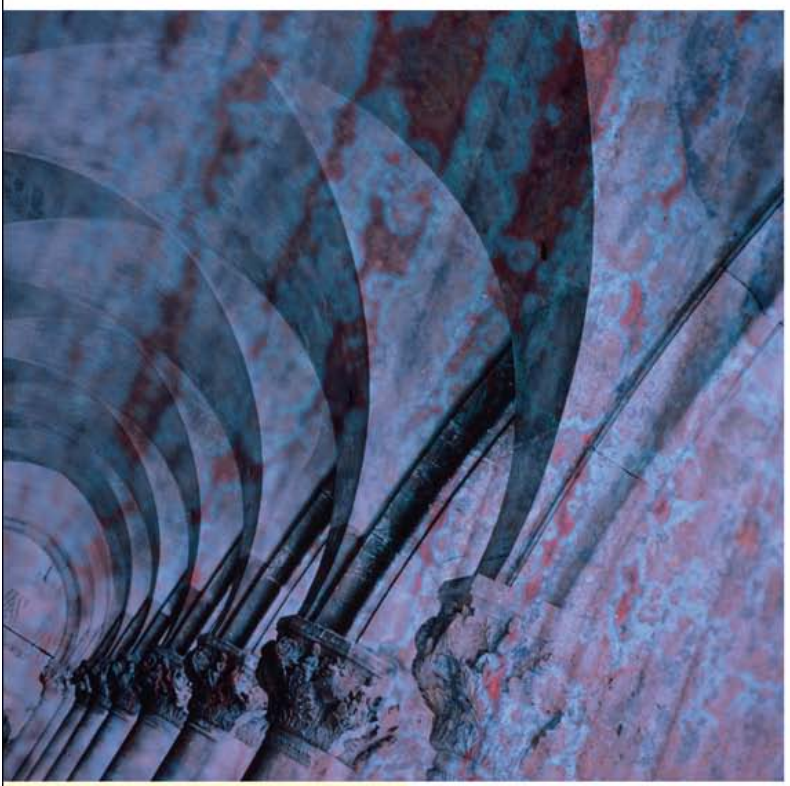
"The MindShare *PCI Express System Architecture* book is expertly aimed at increasing engineer's knowledge of the PCIe specification, leading to increased productivity and time to market."

—David Churchill | Agilent Technologies

PCI Express System Architecture



MINDSHARE

BRINGING LIFE TO KNOWLEDGE

Ravi Budruk
Don Anderson
Tom Shanley

Edited by Joe Winkles

PC SYSTEM ARCHITECTURE SERIES

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

world-class technical training

Are your company's technical training needs being addressed in the most effective manner?

MindShare has over 25 years experience in conducting technical training on cutting-edge technologies. We understand the challenges companies have when searching for quality, effective training which reduces the students' time away from work and provides cost-effective alternatives. MindShare offers many flexible solutions to meet those needs. Our courses are taught by highly-skilled, enthusiastic, knowledgeable and experienced instructors. We bring life to knowledge through a wide variety of learning methods and delivery options.

training that fits your needs

MindShare recognizes and addresses your company's technical training issues with:

- Scalable cost training
- Just-in-time training
- Training in a classroom, at your cubicle or home office
- Customizable training options
- Overview and advanced topic courses
- Reducing time away from work
- Training delivered effectively globally
- Concurrently delivered multiple-site training

MindShare training courses expand your technical skillset

- PCI Express 2.0®
- Intel Core 2 Processor Architecture
- AMD Opteron Processor Architecture
- Intel 64 and IA-32 Software Architecture
- Intel PC and Chipset Architecture
- PC Virtualization
- USB 2.0
- Wireless USB
- Serial ATA (SATA)
- Serial Attached SCSI (SAS)
- DDR2/DDR3 DRAM Technology
- PC BIOS Firmware
- High-Speed Design
- Windows Internals and Drivers
- Linux Fundamentals

... and many more.

All courses can be customized to meet your group's needs. Detailed course outlines can be found at **www.mindshare.com**

bringing life to knowledge.

real-world tech training put into practice worldwide

*PCI Express ® is a registered trademark of the PCISIG

MindShare Learning Options

MindShare Classroom

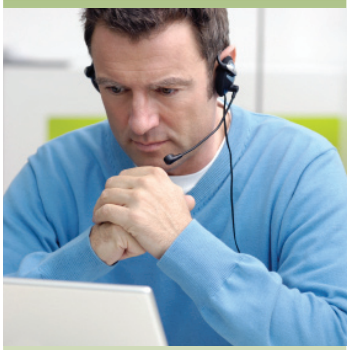
In-House Training

Public Training

Classroom Training

Invite MindShare to train you in-house, or sign-up to attend one of our many public classes held throughout the year and around the world. No more boring classes, the 'MindShare Experience' is sure to keep you engaged.

MindShare Virtual Classroom



Virtual In-House Training

Virtual Public Training

Virtual Classroom Training

The majority of our courses live over the web in an interactive environment with WebEx and a phone bridge. We deliver training cost-effectively across multiple sites and time zones. Imagine being trained in your cubicle or home office and avoiding the hassle of travel. Contact us to attend one of our public virtual classes.

MindShare eLearning

Intro eLearning Modules

Comprehensive eLearning Modules

eLearning Module Training

MindShare is also an eLearning company. Our growing list of interactive eLearning modules include:

- **Intro to Virtualization Technology**
- **Intro to IO Virtualization**
- **Intro to PCI Express 2.0 Updates**
- **PCI Express 2.0**
- **USB 2.0**
- **AMD Opteron Processor Architecture**
- **Virtualization Technology**

...and more

MindShare Press

Books

eBooks

MindShare Press

Purchase our books and eBooks or publish your own content through us. MindShare has authored over 25 books and the list is growing. Let us help make your book project a successful one.

Engage MindShare

Have knowledge that you want to bring to life? MindShare will work with you to "Bring Your Knowledge to Life." Engage us to transform your knowledge and design courses that can be delivered in classroom or virtual classroom settings, create online eLearning modules, or publish a book that you author.

We are proud to be the preferred training provider at an extensive list of clients that include:

ADAPTEC • AMD • AGILENT TECHNOLOGIES • APPLE • BROADCOM • CADENCE • CRAY • CISCO • DELL • FREESCALE
GENERAL DYNAMICS • HP • IBM • KODAK • LSI LOGIC • MOTOROLA • MICROSOFT • NASA • NATIONAL SEMICONDUCTOR
NETAPP • NOKIA • NVIDIA • PLX TECHNOLOGY • QLOGIC • SIEMENS • SUN MICROSYSTEMS SYNOPSYS • TI • UNISYS

MINDSHARE
BRINGING LIFE TO KNOWLEDGE

4285 SLASH PINE DRIVE COLORADO SPRINGS, CO 80908 USA
M 1.602.617.1123 O 1.800.633.1440 ravi@mindshare.com

www.mindshare.com

PCI Express System Architecture

MINDSHARE, INC.

Ravi Budruk
Don Anderson
Tom Shanley

Technical Edit by Joe Winkles

ADDISON-WESLEY DEVELOPER'S PRESS

Boston • San Francisco • New York • Toronto
Montreal • London • Munich • Paris • Madrid • Sydney
Cape Town • Tokyo • Singapore • Mexico City

Library of Congress Cataloging-in-Publication Data

ISBN: 0-321-15630-7

Sponsoring Editor:
Project Manager:
Cover Design:
Set in 10 point Palatino by MindShare, Inc.

1 2 3 4 5 6 7 8 9-MA-999897
9th Printing, April, 2008

Addison-Wesley books available for bulk purchases by corporations, institutions, and other organizations. For more information please contact the Corporate, Government, and Special Sales Department at (800) 238-9682.

Find A-W Developer's Press on the World-Wide Web at:
http://www.awl.com/devpress/

About This Book

Part One: The Big Picture

Chapter 1: Architectural Perspective

Chapter 2: Architecture Overview

Chapter 4: Packet-Based Transactions

Chapter 5: ACK/NAK Protocol

Chapter 6: QoS/TCs/VCs and Arbitration

Chapter 7: Flow Control

Chapter 8: Transaction Ordering

Chapter 9: Interrupts

Chapter 10: Error Detection and Handling

Part Three: The Physical Layer

Chapter 11: Physical Layer Logic

Chapter 12: Electrical Physical Layer

Chapter 13: System Reset

Chapter 14: Link Initialization & Training

Part Four: Power-Related Topics

Chapter 15: Power Budgeting

Chapter 16: Power Management

Part Five: Optional Topics

Chapter 17: Hot Plug

Chapter 18: Add-in Cards and Connectors

Part Six: PCI Express Configuration

Chapter 19: Configuration Overview

Chapter 20: Configuration Mechanisms

Chapter 21: PCI Express Enumeration

Chapter 22: PCI Compatible Configuration Registers

Chapter 23: Expansion ROMs

Chapter 24: Express-Specific Configuration Registers

Appendices

1 Architectural Perspective

This Chapter

This chapter describes performance advantages and key features of the PCI Express (PCI-XP) Link. To highlight these advantages, this chapter describes performance characteristics and features of predecessor buses such as PCI and PCI-X buses with the goal of discussing the evolution of PCI Express from these predecessor buses. The reader will be able to compare and contrast features and performance points of PCI, PCI-X and PCI Express buses. The key features of a PCI Express system are described. In addition, the chapter describes some examples of PCI Express system topologies.

The Next Chapter

The next chapter describes in further detail the features of the PCI Express bus. It describes the layered architecture of a device design while providing a brief functional description of each layer. The chapter provides an overview of packet formation at a transmitter device, the transmission and reception of the packet over the PCI Express Link and packet decode at a receiver device.

Introduction To PCI Express

PCI Express is the third generation high performance I/O bus used to interconnect peripheral devices in applications such as computing and communication platforms. The first generation buses include the ISA, EISA, VESA, and Micro Channel buses, while the second generation buses include PCI, AGP, and PCI-X. PCI Express is an all encompassing I/O device interconnect bus that has applications in the mobile, desktop, workstation, server, embedded computing and communication platforms.

The Role of the Original PCI Solution

Don't Throw Away What is Good! Keep It

The PCI Express architects have carried forward the most beneficial features from previous generation bus architectures and have also taken advantages of new developments in computer architecture.

For example, PCI Express employs the same usage model and load-store communication model as PCI and PCI-X. PCI Express supports familiar transactions such as memory read/write, IO read/write and configuration read/write transactions. The memory, IO and configuration address space model is the same as PCI and PCI-X address spaces. By maintaining the address space model, existing OSs and driver software will run in a PCI Express system without any modifications. In other words, PCI Express is software backwards compatible with PCI and PCI-X systems. In fact, a PCI Express system will boot an existing OS with no changes to current drivers and application programs. Even PCI/ACPI power management software will still run.

Like predecessor buses, PCI Express supports chip-to-chip interconnect and board-to-board interconnect via cards and connectors. The connector and card structure are similar to PCI and PCI-X connectors and cards. A PCI Express motherboard will have a similar form factor to existing FR4 ATX motherboards which is encased in the familiar PC package.

Make Improvements for the Future

To improve bus performance, reduce overall system cost and take advantage of new developments in computer design, the PCI Express architecture had to be significantly re-designed from its predecessor buses. PCI and PCI-X buses are multi-drop parallel interconnect buses in which many devices share one bus.

PCI Express on the other hand implements a serial, point-to-point type interconnect for communication between two devices. Multiple PCI Express devices are interconnected via the use of switches which means one can practically connect a large number of devices together in a system. A point-to-point interconnect implies limited electrical load on the link allowing transmission and reception frequencies to scale to much higher numbers. Currently PCI Express transmission and reception data rate is 2.5 Gbits/sec. A serial interconnect between two devices results in fewer pins per device package which reduces PCI Express chip and board design cost and reduces board design complexity. PCI Express performance is also highly scalable. This is achieved by implement-

ing scalable numbers for pins and signal Lanes per interconnect based on communication performance requirements for that interconnect.

PCI Express implements switch-based technology to interconnect a large number of devices. Communication over the serial interconnect is accomplished using a packet-based communication protocol. Quality Of Service (QoS) features provide differentiated transmission performance for different applications. Hot Plug/Hot Swap support enables "always-on" systems. Advanced power management features allow one to design for low power mobile applications. RAS (Reliable, Available, Serviceable) error handling features make PCI Express suitable for robust high-end server applications. Hot plug, power management, error handling and interrupt signaling are accomplished in-band using packet based messaging rather than side-band signals. This keeps the device pin count low and reduces system cost.

The configuration address space available per function is extended to 4KB, allowing designers to define additional registers. However, new software is required to access this extended configuration register space.

Looking into the Future

In the future, PCI Express communication frequencies are expected to double and quadruple to 5 Gbits/sec and 10 Gbits/sec. Taking advantage of these frequencies will require Physical Layer re-design of a device with no changes necessary to the higher layers of the device design.

Additional mechanical form factors are expected. Support for a Server IO Module, Newcard (PC Card style), and Cable form factors are expected.

Predecessor Buses Compared

In an effort to compare and contrast features of predecessor buses, the next section of this chapter describes some of the key features of IO bus architectures defined by the PCI Special Interest Group (PCISIG). These buses, shown in Table 1-1 on page 12, include the PCI 33 MHz bus, PCI- 66 MHz bus, PCI-X 66 MHz/133 MHz buses, PCI-X 266/533 MHz buses and finally PCI Express.

Table 1-1: Bus Specifications and Release Dates

Bus Type	Specification Release	Date of Release
PCI 33 MHz	2.0	1993
PCI 66 MHz	2.1	1995
PCI-X 66 MHz and 133 MHz	1.0	1999
PCI-X 266 MHz and 533 MHz	2.0	Q1, 2002
PCI Express	1.0	Q2, 2002

Author's Disclaimer

In comparing these buses, it is not the authors' intention to suggest that any one bus is better than any other bus. Each bus architecture has its advantages and disadvantages. After evaluating the features of each bus architecture, a particular bus architecture may turn out to be more suitable for a specific application than another bus architecture. For example, it is the system designers responsibility to determine whether to implement a PCI-X bus or PCI Express for the I/O interconnect in a high-end server design. Our goal in this chapter is to document the features of each bus architecture so that the designer can evaluate the various bus architectures.

Bus Performances and Number of Slots Compared

Table 1-2 on page 13 shows the various bus architectures defined by the PCISIG. The table shows the evolution of bus frequencies and bandwidths. As is obvious, increasing bus frequency results in increased bandwidth. However, increasing bus frequency compromises the number of electrical loads or number of connectors allowable on a bus at that frequency. At some point, for a given bus architecture, there is an upper limit beyond which one cannot further increase the bus frequency, hence requiring the definition of a new bus architecture.

Table 1-2: Comparison of Bus Frequency, Bandwidth and Number of Slots

Bus Type	Clock Frequency	Peak Bandwidth *	Number of Card Slots per Bus
PCI 32-bit	33 MHz	133 MBytes/sec	4-5
PCI 32-bit	66 MHz	266 MBytes/sec	1-2
PCI-X 32-bit	66 MHz	266 MBytes/sec	4
PCI-X 32-bit	133 MHz	533 MBytes/sec	1-2
PCI-X 32-bit	266 MHz effective	1066 MBytes/sec	1
PCI-X 32-bit	533 MHz effective	2131 MByte/sec	1
*** Double all these bandwidth numbers for 64-bit bus implementations**			

PCI Express Aggregate Throughput

A PCI Express interconnect that connects two devices together is referred to as a Link. A Link consists of either x1, x2, x4, x8, x12, x16 or x32 signal pairs in each direction. These signals are referred to as Lanes. A designer determines how many Lanes to implement based on the targeted performance benchmark required on a given Link.

Table 1-3 shows aggregate bandwidth numbers for various Link width implementations. As is apparent from this table, the peak bandwidth achievable with PCI Express is significantly higher than any existing bus today.

Let us consider how these bandwidth numbers are calculated. The transmission/reception rate is 2.5 Gbits/sec per Lane per direction. To support a greater degree of robustness during data transmission and reception, each byte of data transmitted is converted into a 10-bit code (via an 8b/10b encoder in the transmitter device). In other words, for every Byte of data to be transmitted, 10-bits of encoded data are actually transmitted. The result is 25% additional overhead to transmit a byte of data. Table 1-3 accounts for this 25% loss in transmission performance.

PCI Express implements a dual-simplex Link which implies that data is transmitted and received simultaneously on a transmit and receive Lane. The aggregate bandwidth assumes simultaneous traffic in both directions.

To obtain the aggregate bandwith numbers in Table 1-3 multiply 2.5 Gbits/sec by 2 (for each direction), then multiply by number of Lanes, and finally divide by 10-bits per Byte (to account for the 8-to-10 bit encoding).

Table 1-3: PCI Express Aggregate Throughput for Various Link Widths

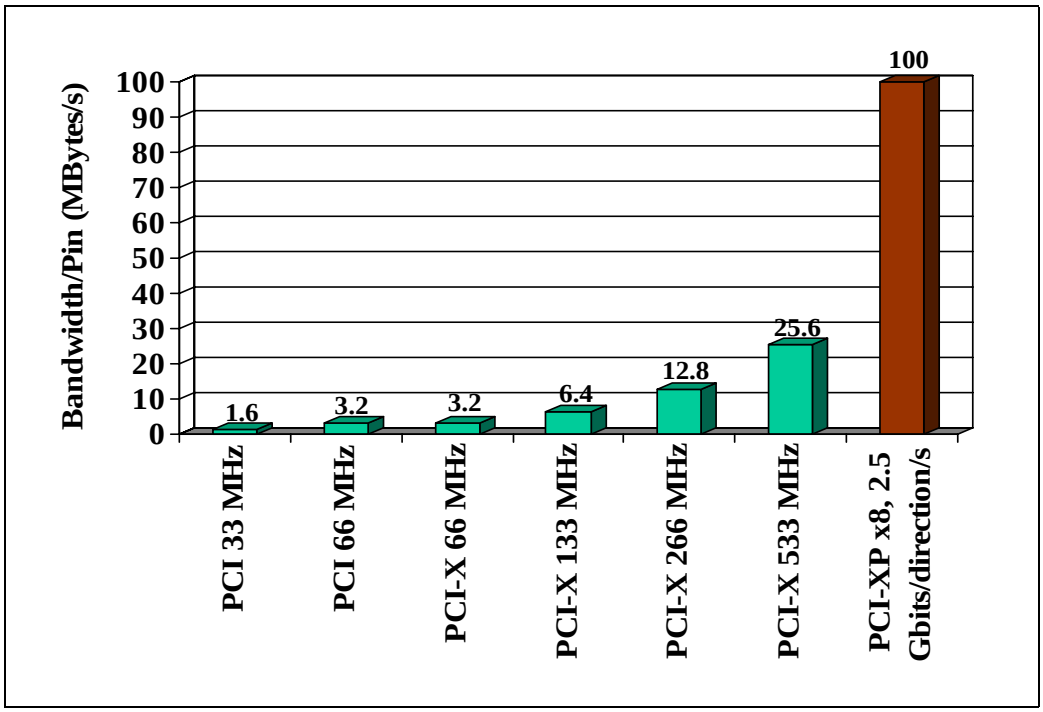
PCI Express Link Width	x1	x2	x4	x8	x12	x16	x32
Aggregate Bandwidth (GBytes/sec)	0.5	1	2	4	6	8	16

Performance Per Pin Compared

As is apparent from Figure 1-1, PCI Express achieves the highest bandwidth per pin. This results in a device package with fewer pins and a motherboard implementation with few wires and hence overall reduced system cost per unit bandwidth.

In Figure 1-1, the first 7 bars are associated with PCI and PCI-X buses where we assume 84 pins per device. This includes 46 signal pins, interrupt and power management pins, error pins and the remainder are power and ground pins. The last bar associated with a x8 PCI Express Link assumes 40 pins per device which include 32 signal lines (8 differential pairs per direction) and the rest are power and ground pins.

Figure 1-1: Comparison of Performance Per Pin for Various Buses



I/O Bus Architecture Perspective

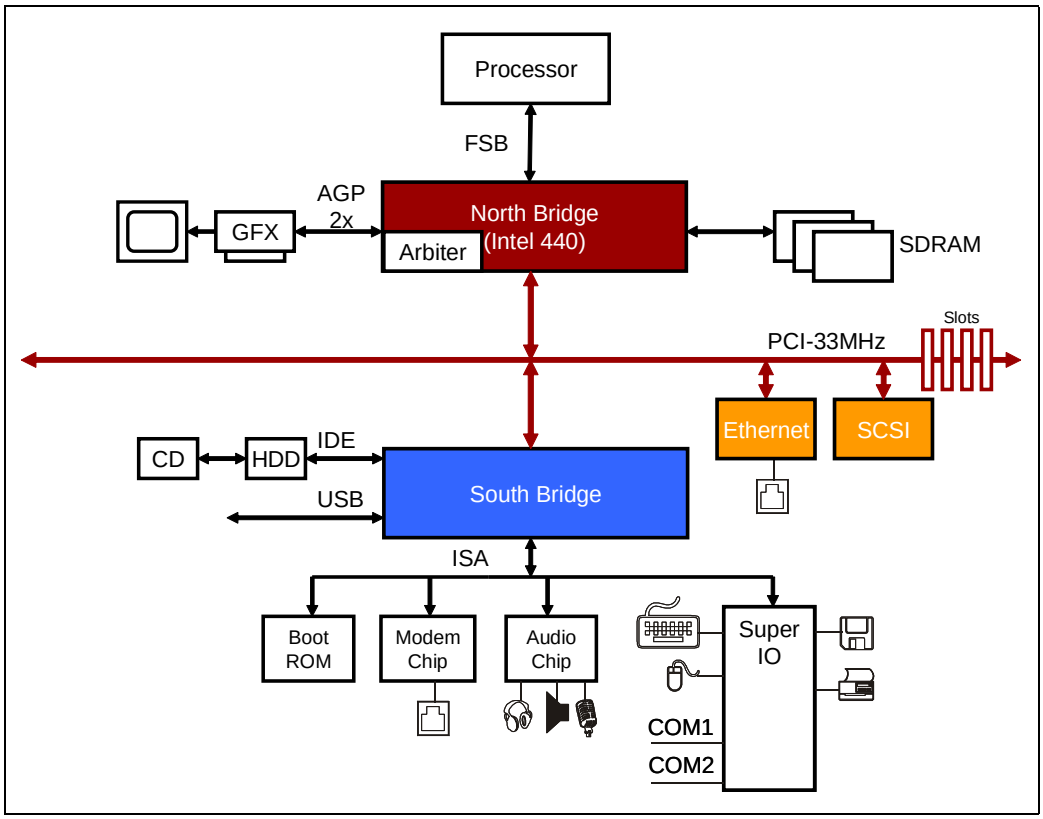
33 MHz PCI Bus Based System

Figure 1-2 on page 17 is a 33 MHz PCI bus based system. The PCI system consists of a Host (CPU) bus-to-PCI bus bridge, also referred to as the North bridge. Associated with the North bridge is the system memory bus, graphics (AGP) bus, and a 33 MHz PCI bus. I/O devices share the PCI bus and are connected to it in a multi-drop fashion. These devices are either connected directly to the PCI bus on the motherboard or by way of a peripheral card plugged into a connector on the bus. Devices connected directly to the motherboard consume one electrical load while connectors are accounted for as 2 loads. A South bridge bridges the PCI bus to the ISA bus where slower, lower performance peripherals exist. Associated with the south bridge is a USB and IDE bus. A CD or hard disk is associated with the IDE bus. The South bridge contains an interrupt controller (not shown) to which interrupt signals from PCI devices are connected. The interrupt controller is connected to the CPU via an INTR signal or an APIC bus. The South bridge is the central resource that provides the source of reset, reference clock, and error reporting signals. Boot ROM exists on the ISA bus along with a Super IO chip, which includes keyboard, mouse, floppy disk controller and serial/parallel bus controllers. The PCI bus arbiter logic is included in the North bridge.

Figure 1-3 on page 18 represents a typical PCI bus cycle. The PCI bus clock is 33 MHz. The address bus width is 32-bits (4GB memory address space), although PCI optionally supports 64-bit address bus. The data bus width is implemented as either 32-bits or 64-bits depending on bus performance requirement. The address and data bus signals are multiplexed on the same pins (AD bus) to reduce pin count. Command signals (C/BE#) encode the transaction type of the bus cycle that master devices initiate. PCI supports 12 transaction types that include memory, IO, and configuration read/write bus cycles. Control signals such as FRAME#, DEVSEL#, TRDY#, IRDY#, STOP# are handshake signals used during bus cycles. Finally, the PCI bus consists of a few optional error related signals, interrupt signals and power management signals. A PCI master device implements a minimum of 49 signals.

Any PCI master device that wishes to initiate a bus cycle first arbitrates for use of the PCI bus by asserting a request (REQ#) to the arbiter in the North bridge. After receiving a grant (GNT#) from the arbiter and checking that the bus is idle, the master device can start a bus cycle.

Figure 1-2: 33 MHz PCI Bus Based Platform



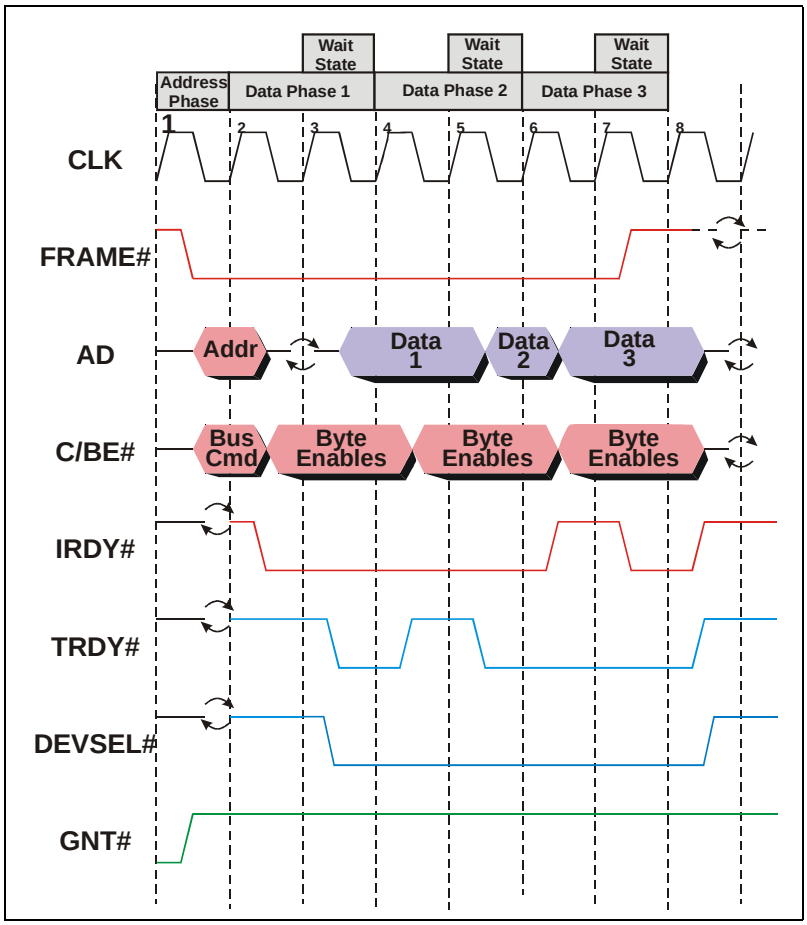
Electrical Load Limit of a 33 MHz PCI Bus

The PCI specification theoretically supports 32 devices per PCI bus. This means that PCI enumeration software will detect and recognize up to 32 devices per bus. However, as a rule of thumb, a PCI bus can support a maximum of 10-12 electrical loads (devices) at 33 MHz. PCI implements a static clocking protocol with a clock period of 30 ns at 33 MHz.

PCI implements reflected-wave switching signal drivers. The driver drives a half signal swing signal on the rising edge of PCI clock. The signal propagates down the PCI bus transmission line and is reflected at the end of the transmission line where there is no termination. The reflection causes the half swing signal to double. The doubled (full signal swing) signal must settle to a steady state

value with sufficient setup time prior to the next rising edge of PCI clock where receiving devices sample the signal. The total time from when a driver drives a signal until the receiver detects a valid signal (including propagation time and reflection delay plus setup time) must be less than the clock period of 30 ns.

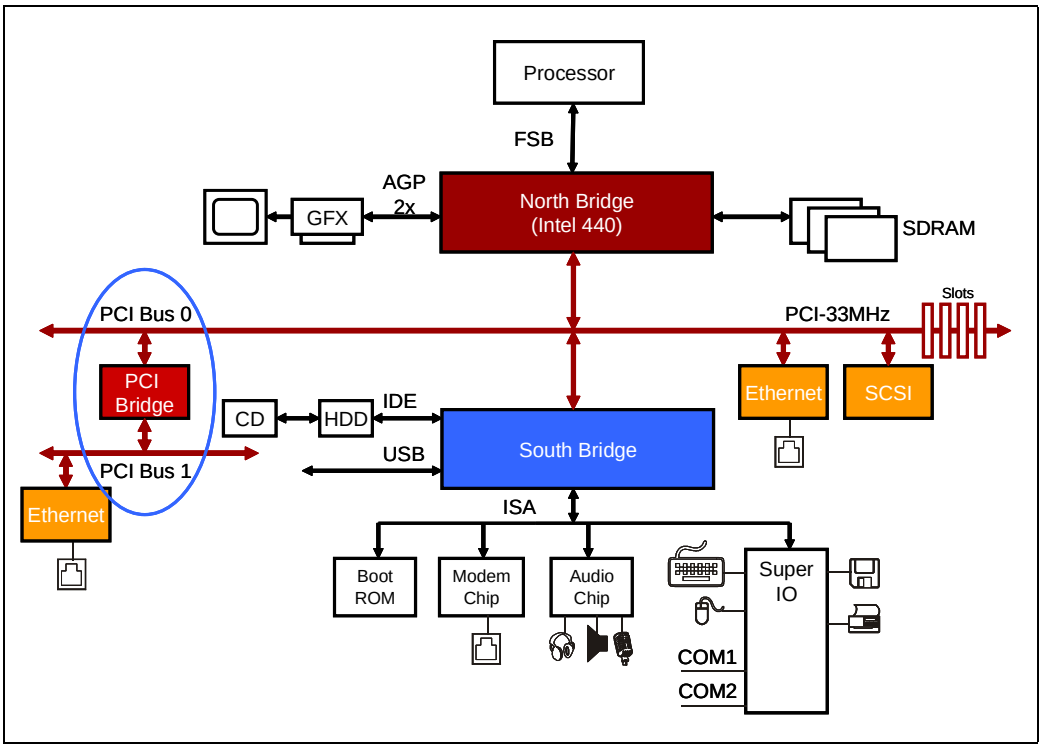
Figure 1-3: Typical PCI Burst Memory Read Bus Cycle



The more electrical loads on a bus, the longer it takes for the signal to propagate and double with sufficient setup to the next rising edge of clock. As mentioned earlier, a 33 MHz PCI bus meets signal timing with no more than 10-12 loads. Connectors on the PCI bus are counted as 2 loads because the connector is accounted for as one load and the peripheral card with a PCI device is the second load. As indicated in Table 1-2 on page 13 a 33 MHz PCI bus can be designed with a maximum of 4-5 connectors.

To connect any more than 10-12 loads in a system requires the implementation of a PCI-to-PCI bridge as shown in Figure 1-4. This permits an additional 10-12 loads to be connected on the secondary PCI bus 1. The PCI specification theoretically supports up to 256 buses in a system. This means that PCI enumeration software will detect and recognize up to 256 PCI bridges per system.

Figure 1-4: 33 MHz PCI Based System Showing Implementation of a PCI-to-PCI Bridge

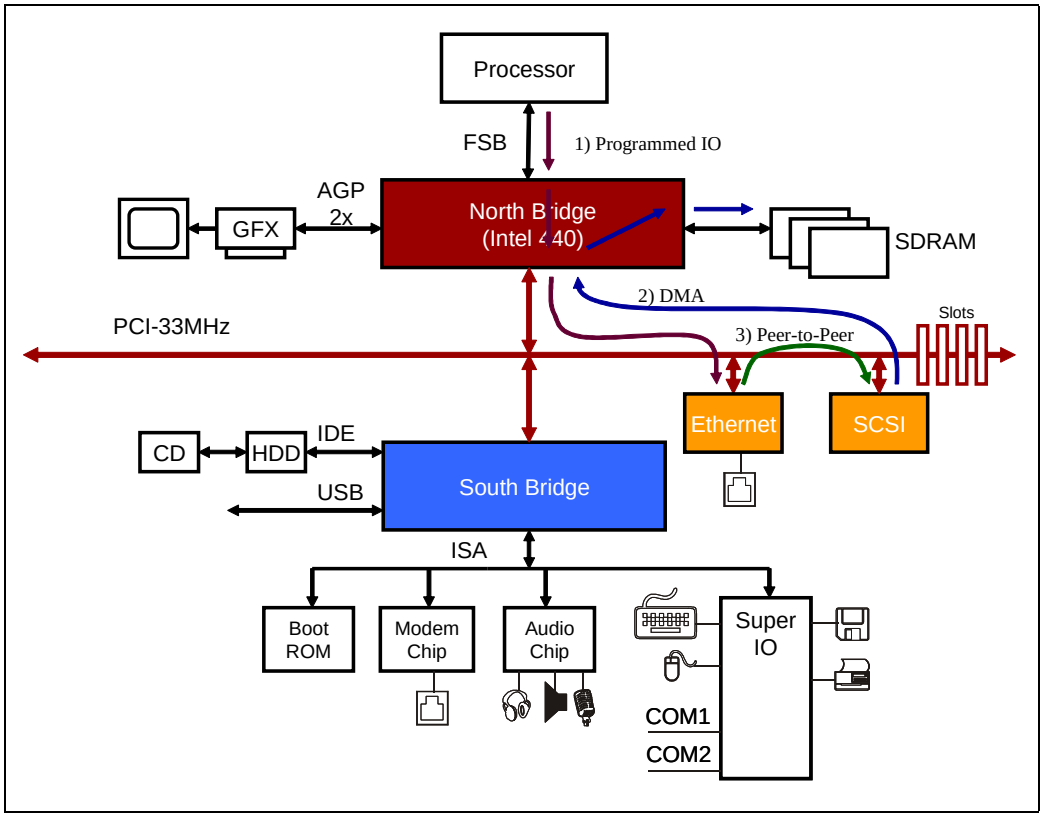


PCI Transaction Model - Programmed IO

Consider an example in which the CPU communicates with a PCI peripheral such as an Ethernet device shown in Figure 1-5. Transaction 1 shown in the figure, which is initiated by the CPU and targets a peripheral device, is referred to as a programmed IO transaction. Software commands the CPU to initiate a memory or IO read/write bus cycle on the host bus targeting an address mapped in a PCI device's address space. The North bridge arbitrates for use of the PCI bus and when it wins ownership of the bus generates a PCI memory or IO read/write bus cycle represented in Figure 1-3 on page 18. During the first clock of this bus cycle (known as the address phase), all target devices decode

the address. One target (the Ethernet device in this example) decodes the address and claims the transaction. The master (North bridge in this case) communicates with the claiming target (Ethernet controller). Data is transferred between master and target in subsequent clocks after the address phase of the bus cycle. Either 4 bytes or 8 bytes of data are transferred per clock tick depending on the PCI bus width. The bus cycle is referred to as a burst bus cycle if data is transferred back-to-back between master and target during multiple data phases of that bus cycle. Burst bus cycles result in the most efficient use of PCI bus bandwidth.

Figure 1-5: PCI Transaction Model



At 33 MHz and the bus width of 32-bits (4 Bytes), peak bandwidth achievable is 4 Bytes x 33 MHz = 133 MBytes/sec. Peak bandwidth on a 64-bit bus is 266 Mbytes/sec. See Table 1-2 on page 13.

2 Architecture Overview

Previous Chapter

The previous chapter described performance advantages and key features of the PCI Express (PCI-XP) Link. To highlight these advantages, the chapter described performance characteristics and features of predecessor buses such as PCI and PCI-X buses with the goal of discussing the evolution of PCI Express from these predecessor buses. It compared and contrasted features and performance points of PCI, PCI-X and PCI Express buses. The key features of a PCI Express system were described. The chapter in addition described some examples of PCI Express system topologies.

This Chapter

This chapter is an introduction to the PCI Express data transfer protocol. It describes the layered approach to PCI Express device design while describing the function of each device layer. Packet types employed in accomplishing data transfers are described without getting into packet content details. Finally, this chapter outlines the process of a requester initiating a transaction such as a memory read to read data from a completer across a Link.

The Next Chapter

The next chapter describes how packets are routed through a PCI Express fabric consisting of switches. Packets are routed based on a memory address, IO address, device ID or implicitly.

Introduction to PCI Express Transactions

PCI Express employs packets to accomplish data transfers between devices. A root complex can communicate with an endpoint. An endpoint can communicate with a root complex. An endpoint can communicate with another endpoint. Communication involves the transmission and reception of packets called Transaction Layer packets (TLPs).

PCI Express transactions can be grouped into four categories:

1) memory, 2) IO, 3) configuration, and 4) message transactions. Memory, IO and configuration transactions are supported in PCI and PCI-X architectures, but the message transaction is new to PCI Express. **Transactions** are defined as a series of one or more packet transmissions required to complete an information transfer between a requester and a completer. Table 2-1 is a more detailed list of transactions. These transactions can be categorized into non-posted transactions and posted transactions.

Table 2-1: PCI Express Non-Posted and Posted Transactions

Transaction Type	Non-Posted or Posted
Memory Read	Non-Posted
Memory Write	Posted
Memory Read Lock	Non-Posted
IO Read	Non-Posted
IO Write	Non-Posted
Configuration Read (Type 0 and Type 1)	Non-Posted
Configuration Write (Type 0 and Type 1)	Non-Posted
Message	Posted

For Non-posted transactions, a requester transmits a TLP request packet to a completer. At a later time, the completer returns a TLP completion packet back to the requester. Non-posted transactions are handled as split transactions similar to the PCI-X split transaction model described on page 37 in Chapter 1. The purpose of the completion TLP is to confirm to the requester that the completer has received the request TLP. In addition, non-posted read transactions contain data in the completion TLP. Non-Posted write transactions contain data in the write request TLP.

For Posted transactions, a requester transmits a TLP request packet to a completer. The completer however does NOT return a completion TLP back to the requester. Posted transactions are optimized for best performance in completing the transaction at the expense of the requester not having knowledge of successful reception of the request by the completer. Posted transactions may or may not contain data in the request TLP.

PCI Express Transaction Protocol

Table 2-2 lists all of the TLP request and TLP completion packets. These packets are used in the transactions referenced in Table 2-1. Our goal in this section is to describe how these packets are used to complete transactions at a system level and not to describe the packet routing through the PCI Express fabric nor to describe packet contents in any detail.

Table 2-2: PCI Express TLP Packet Types

TLP Packet Types	Abbreviated Name
Memory Read Request	MRd
Memory Read Request - Locked access	MRdLk
Memory Write Request	MWr
IO Read	IORd
IO Write	IOWr
Configuration Read (Type 0 and Type 1)	CfgRd0, CfgRd1
Configuration Write (Type 0 and Type 1)	CfgWr0, CfgWr1
Message Request without Data	Msg
Message Request with Data	MsgD
Completion without Data	Cpl
Completion with Data	CplD
Completion without Data - associated with Locked Memory Read Requests	CplLk
Completion with Data - associated with Locked Memory Read Requests	CplDLk

Non-Posted Read Transactions

Figure 2-1 shows the packets transmitted by a requester and completer to complete a non-posted read transaction. To complete this transfer, a requester transmits a non-posted read request TLP to a completer it intends to read data from. Non-posted read request TLPs include memory read request (MRd), IO read request (IORd), and configuration read request type 0 or type 1 (CfgRd0, CfgRd1) TLPs. Requesters may be root complex or endpoint devices (endpoints do not initiate configuration read/write requests however).

The request TLP is routed through the fabric of switches using information in the header portion of the TLP. The packet makes its way to a targeted completer. The completer can be a root complex, switches, bridges or endpoints.

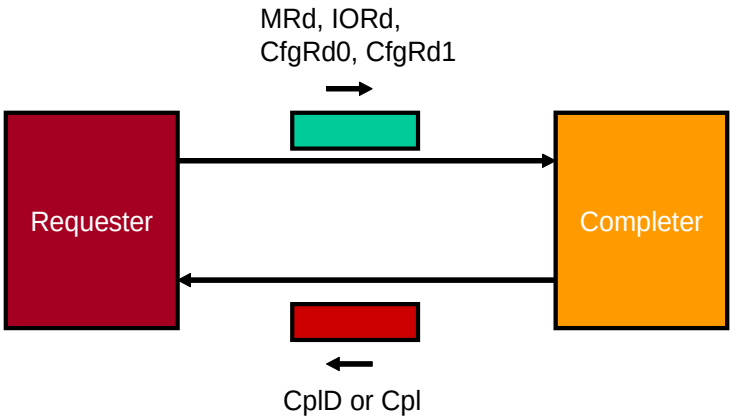
When the completer receives the packet and decodes its contents, it gathers the amount of data specified in the request from the targeted address. The completer creates a single completion TLP or multiple completion TLPs with data (CplD) and sends it back to the requester. The completer can return up to 4 KBytes of data per CplD packet.

The completion packet contains routing information necessary to route the packet back to the requester. This completion packet travels through the same path and hierarchy of switches as the request packet.

Requesters uses a tag field in the completion to associate it with a request TLP of the same tag value it transmitted earlier. Use of a tag in the request and completion TLPs allows a requester to manage multiple outstanding transactions.

If a completer is unable to obtain requested data as a result of an error, it returns a completion packet without data (Cpl) and an error status indication. The requester determines how to handle the error at the software layer.

Figure 2-1: Non-Posted Read Transaction Protocol



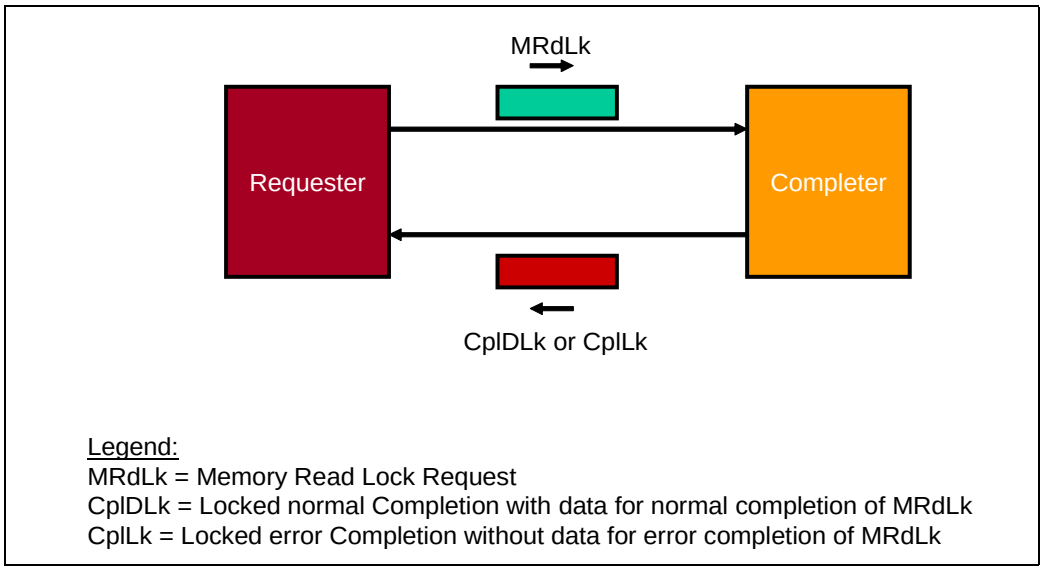
Legend:
MRd = Memory Read Request
IORd = IO Read Request
CfgRd0 = Type 0 Configuration Read Request
CfgRd1 = Type 1 Configuration Read Request
CplD = Completion with data for normal completion of MRd, IORd, CfgRd0, CfgRd1
Cpl = Completion without data for error completion of MRd, IORd, CfgRd0, CfgRd1

Non-Posted Read Transaction for Locked Requests

Figure 2-2 on page 60 shows packets transmitted by a requester and completer to complete a non-posted locked read transaction. To complete this transfer, a requester transmits a memory read locked request (MRdLk) TLP. The requester can only be a root complex which initiates a locked request on the behalf of the CPU. Endpoints are not allowed to initiate locked requests.

The locked memory read request TLP is routed downstream through the fabric of switches using information in the header portion of the TLP. The packet makes its way to a targeted completer. The completer can only be a legacy end-point. The entire path from root complex to the endpoint (for TCs that map to VC0) is locked including the ingress and egress port of switches in the pathway.

Figure 2-2: Non-Posted Locked Read Transaction Protocol



When the completer receives the packet and decodes its contents, it gathers the amount of data specified in the request from the targeted address. The completer creates one or more locked completion TLP with data (CplDLk) along with a completion status. The completion is sent back to the root complex requester via the path and hierarchy of switches as the original request.

The CplDLk packet contains routing information necessary to route the packet back to the requester. Requesters uses a tag field in the completion to associate it with a request TLP of the same tag value it transmitted earlier. Use of a tag in the request and completion TLPs allows a requester to manage multiple outstanding transactions.

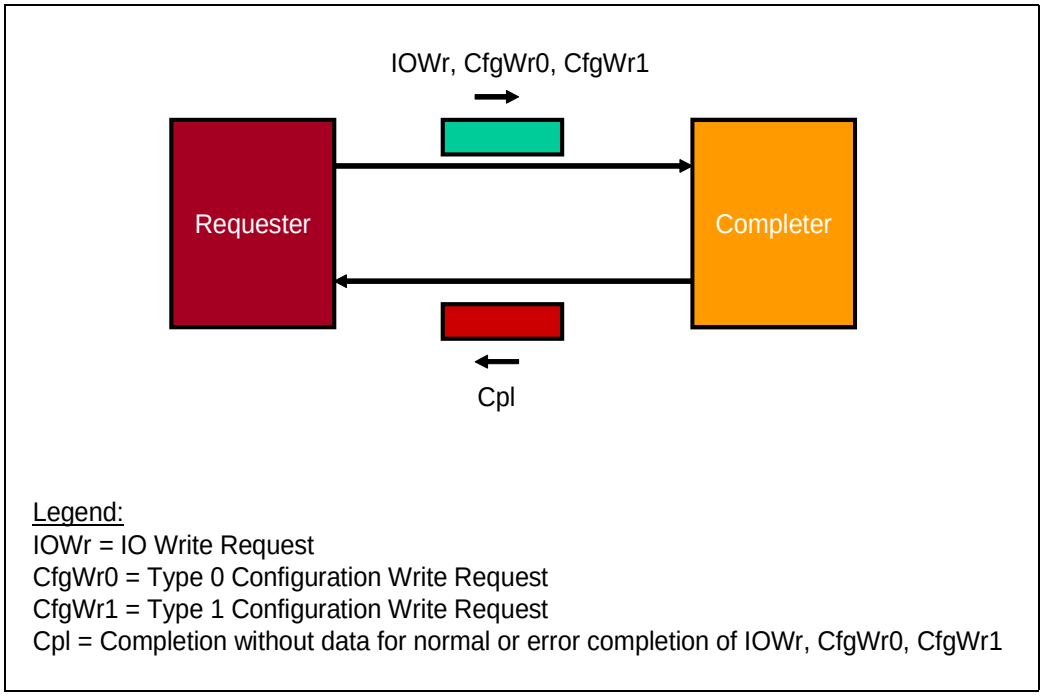
If the completer is unable to obtain the requested data as a result of an error, it returns a completion packet without data (CplLk) and an error status indication within the packet. The requester who receives the error notification via the CplLk TLP must assume that atomicity of the lock is no longer guaranteed and thus determine how to handle the error at the software layer.

The path from requester to completer remains locked until the requester at a later time transmits an unlock message to the completer. The path and ingress/egress ports of a switch that the unlock message passes through are unlocked.

Non-Posted Write Transactions

Figure 2-3 on page 61 shows the packets transmitted by a requester and completer to complete a non-posted write transaction. To complete this transfer, a requester transmits a non-posted write request TLP to a completer it intends to write data to. Non-posted write request TLPs include IO write request (IOWr), configuration write request type 0 or type 1 (CfgWr0, CfgWr1) TLPs. Memory write request and message requests are posted requests. Requesters may be a root complex or endpoint device (though not for configuration write requests).

Figure 2-3: Non-Posted Write Transaction Protocol



A request packet with data is routed through the fabric of switches using information in the header of the packet. The packet makes its way to a completer.

When the completer receives the packet and decodes its contents, it accepts the data. The completer creates a single completion packet without data (Cpl) to confirm reception of the write request. This is the purpose of the completion.

The completion packet contains routing information necessary to route the packet back to the requester. This completion packet will propagate through the same hierarchy of switches that the request packet went through before making its way back to the requester. The requester gets confirmation notification that the write request did make its way successfully to the completer.

If the completer is unable to successfully write the data in the request to the final destination or if the write request packet reaches the completer in error, then it returns a completion packet without data (Cpl) but with an error status indication. The requester who receives the error notification via the Cpl TLP determines how to handle the error at the software layer.

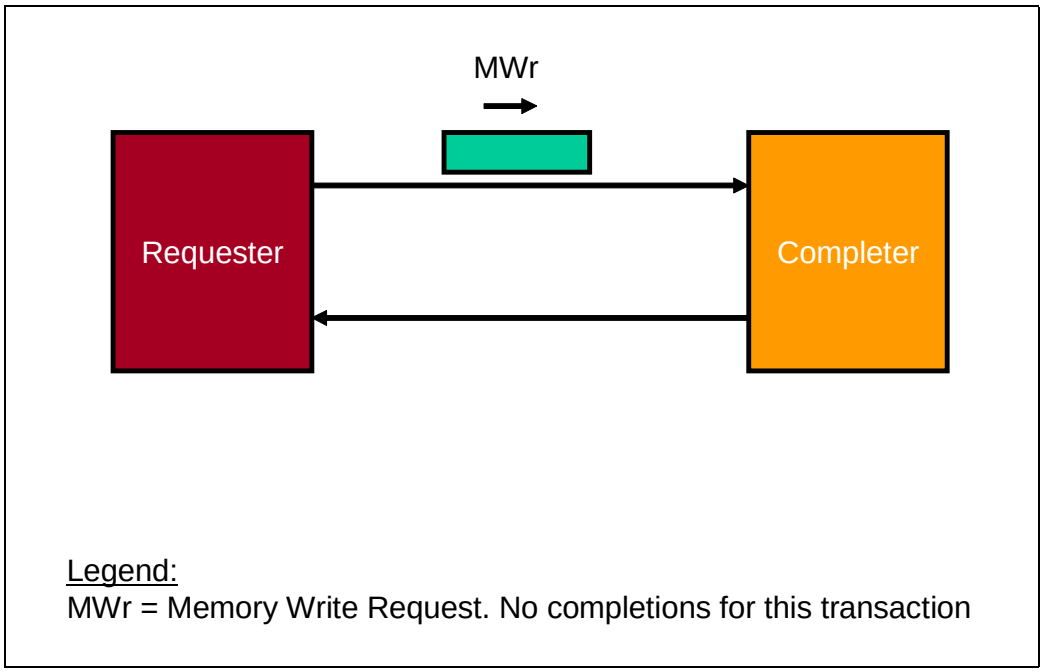
Posted Memory Write Transactions

Memory write requests shown in Figure 2-4 are posted transactions. This implies that the completer returns no completion notification to inform the requester that the memory write request packet has reached its destination successfully. No time is wasted in returning a completion, thus back-to-back posted writes complete with higher performance relative to non-posted transactions.

The write request packet which contains data is routed through the fabric of switches using information in the header portion of the packet. The packet makes its way to a completer. The completer accepts the specified amount of data within the packet. Transaction over.

If the write request is received by the completer in error, or is unable to write the posted write data to the final destination due to an internal error, the requester is not informed via the hardware protocol. The completer could log an error and generate an error message notification to the root complex. Error handling software manages the error.

Figure 2-4: Posted Memory Write Transaction Protocol



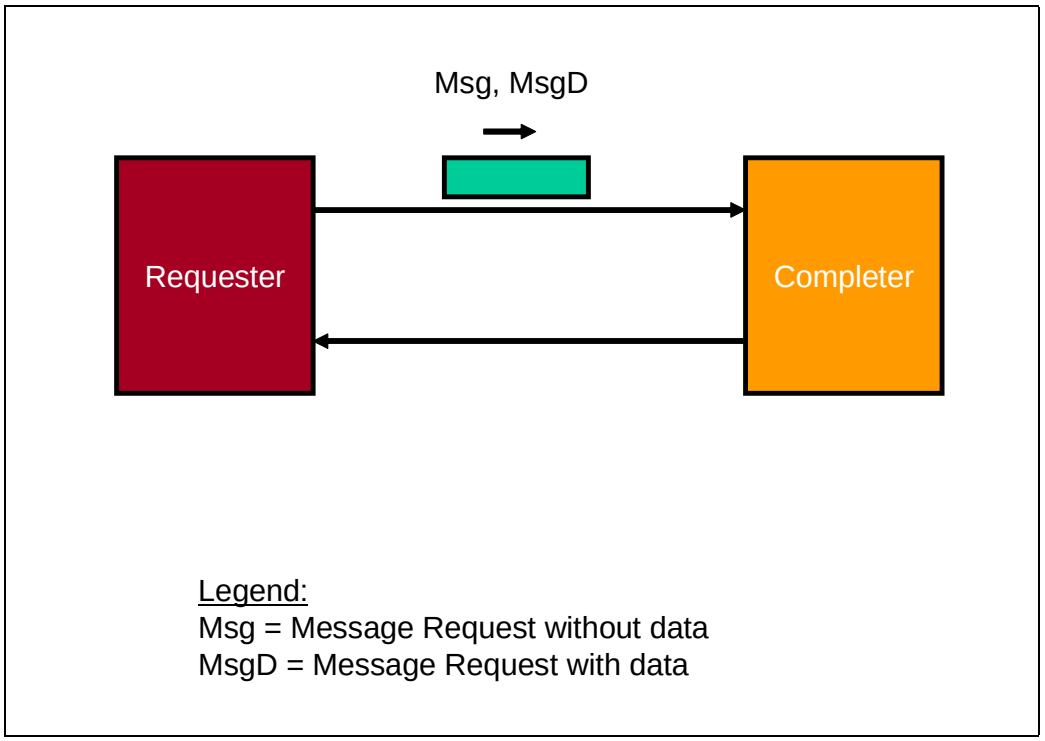
Posted Message Transactions

Message requests are also posted transactions as pictured in Figure 2-5 on page 64. There are two categories of message request TLPs, Msg and MsgD. Some message requests propagate from requester to completer, some are broadcast requests from the root complex to all endpoints, some are transmitted by an endpoint to the root complex. Message packets may be routed to completer(s) based on the message's address, device ID or routed implicitly. Message request routing is covered in Chapter 3.

The completer accepts any data that may be contained in the packet (if the packet is MsgD) and/or performs the task specified by the message.

Message request support eliminates the need for side-band signals in a PCI Express system. They are used for PCI style legacy interrupt signaling, power management protocol, error signaling, unlocking a path in the PCI Express fabric, slot power support, hot plug protocol, and vender defined purposes.

Figure 2-5: Posted Message Transaction Protocol



Some Examples of Transactions

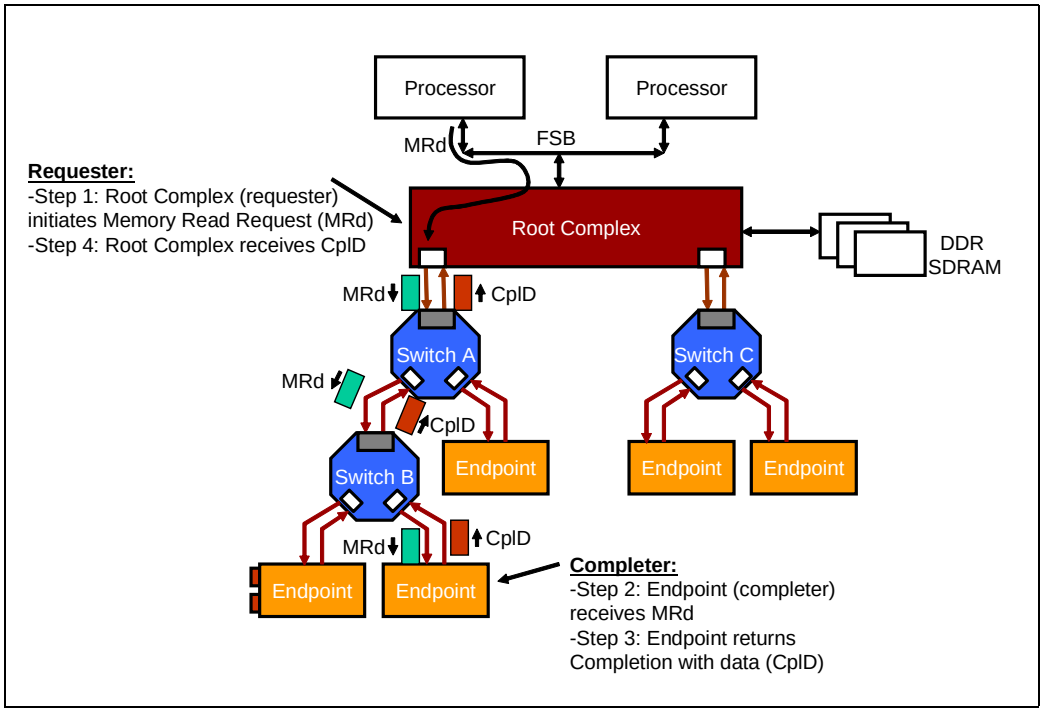
This section describes a few transaction examples showing packets transmitted between requester and completer to accomplish a transaction. The examples consist of a memory read, IO write, and Memory write.

Memory Read Originated by CPU, Targeting an Endpoint

Figure 2-6 shows an example of packet routing associated with completing a memory read transaction. The root complex on the behalf of the CPU initiates a non-posted memory read from the completer endpoint shown. The root complex transmits an MRd packet which contains amongst other fields, an address, TLP type, requester ID (of the root complex) and length of transfer (in double-words) field. Switch A which is a 3 port switch receives the packet on its

upstream port. The switch logically appears like a 3 virtual bridge device connected by an internal bus. The logical bridges within the switch contain memory and IO base and limit address registers within their configuration space similar to PCI bridges. The MRd packet address is decoded by the switch and compared with the base/limit address range registers of the two downstream logical bridges. The switch internally forwards the MRd packet from the upstream ingress port to the correct downstream port (the left port in this example). The MRd packet is forwarded to switch B. Switch B decodes the address in a similar manner. Assume the MRd packets is forwarded to the right-hand port so that the completer endpoint receives the MRd packet.

Figure 2-6: Non-Posted Memory Read Originated by CPU and Targeting an Endpoint



The completer decodes the contents of the header within the MRd packet, gathers the requested data and returns a completion packet with data (CplD). The header portion of the completion TLP contains the requester ID copied from the original request TLP. The requester ID is used to route the completion packet back to the root complex.

3 Address Spaces & Transaction Routing

The Previous Chapter

The previous chapter introduced the PCI Express data transfer protocol. It described the layered approach to PCI Express device design while describing the function of each device layer. Packet types employed in accomplishing data transfers were described without getting into packet content details. Finally, this chapter outlined the process of a requester initiating a transaction such as a memory read to read data from a completer across a Link.

This Chapter

This chapter describes the general concepts of PCI Express transaction routing and the mechanisms used by a device in deciding whether to accept, forward, or reject a packet arriving at an ingress port. Because Data Link Layer Packets (DLLPs) and Physical Layer *ordered set* link traffic are never forwarded, the emphasis here is on Transaction Layer Packet (TLP) types and the three routing methods associated with them: address routing, ID routing, and implicit routing. Included is a summary of configuration methods used in PCI Express to set up PCI-compatible plug-and-play addressing within system IO and memory maps, as well as key elements in the PCI Express packet protocol used in making routing decisions.

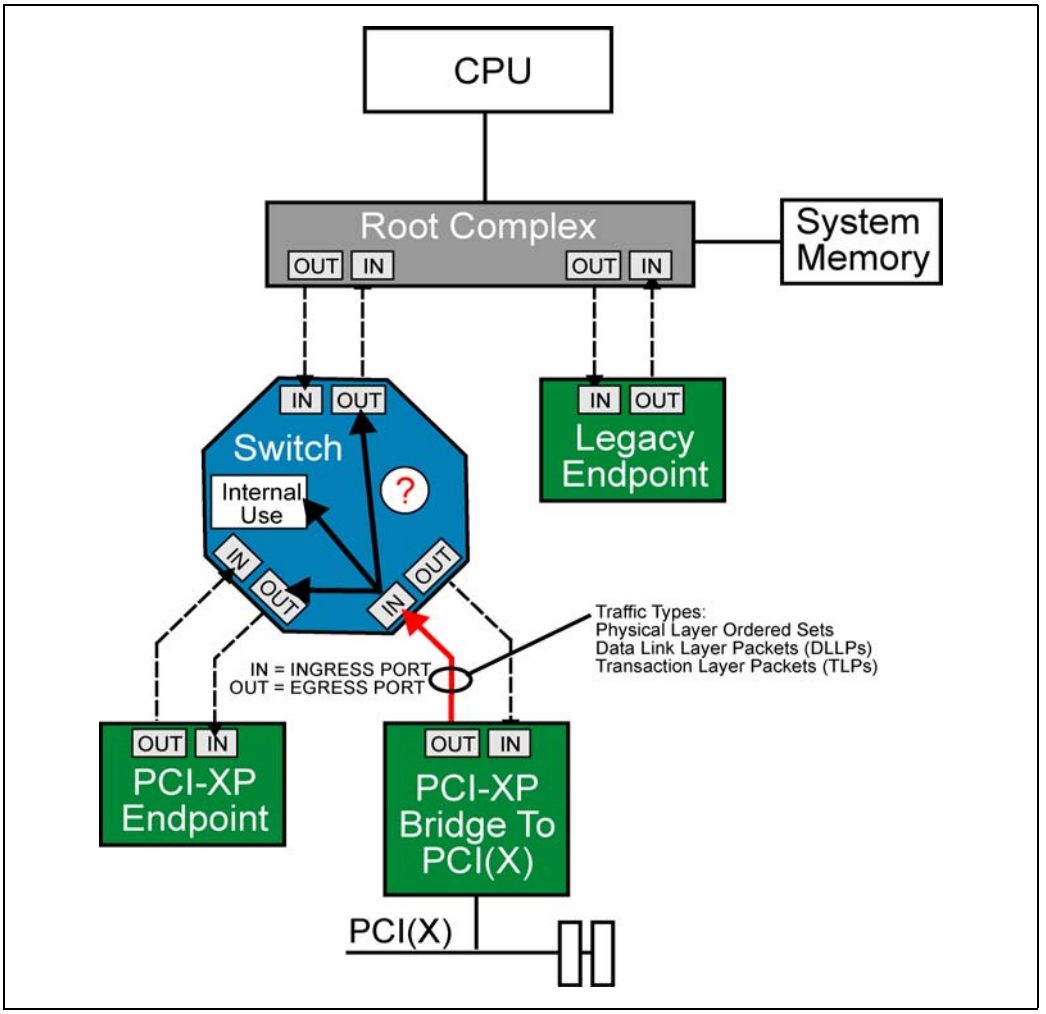
The Next Chapter

The next chapter details the two major classes of packets are *Transaction Layer Packets* (TLPs), and *Data Link Layer Packets* (DLLPs). The use and format of each TLP and DLLP packet type is covered, along with definitions of the field within the packets.

Introduction

Unlike shared-bus architectures such as PCI and PCI-X, where traffic is visible to each device and routing is mainly a concern of bridges, PCI Express devices are dependent on each other to accept traffic or forward it in the direction of the ultimate recipient.

Figure 3-1: Multi-Port PCI Express Devices Have Routing Responsibilities



As illustrated in Figure 3-1 on page 106, a PCI Express topology consists of independent, point-to-point links connecting each device with one or more neighbors. As traffic arrives at the inbound side of a link interface (called the *ingress port*), the device checks for errors, then makes one of three decisions:

1. Accept the traffic and use it internally.
2. Forward the traffic to the appropriate outbound *(egress*) port.
3. Reject the traffic because it is neither the intended target nor an interface to it (note that there are also other reasons why traffic may be rejected)

Receivers Check For Three Types of Link Traffic

Assuming a link is fully operational, the physical layer receiver interface of each device is prepared to monitor the logical idle condition and detect the arrival of the three types of link traffic: Ordered Sets, DLLPs, and TLPs. Using control (K) symbols which accompany the traffic to determine framing boundaries and traffic type, PCI Express devices then make a distinction between traffic which is local to the link vs. traffic which may require routing to other links (e.g. TLPs). Local link traffic, which includes Ordered Sets and Data Link Layer Packets (DLLPs), isn't forwarded and carries no routing information. Transaction Layer Packets (TLPs) can and do move from link to link, using routing information contained in the packet headers.

Multi-port Devices Assume the Routing Burden

It should be apparent in Figure 3-1 on page 106 that devices with multiple PCI Express ports are responsible for handling their own traffic as well as forwarding other traffic between ingress ports and any enabled egress ports. Also note that while peer-peer transaction support is required of switches, it is optional for a multi-port Root Complex. It is up to the system designer to account for peer-to-peer traffic when selecting devices and laying out a motherboard.

Endpoints Have Limited Routing Responsibilities

It should also be apparent in Figure 3-1 on page 106 that endpoint devices have a single link interface and lack the ability to route inbound traffic to other links. For this reason, and because they don't reside on shared busses, endpoints never expect to see ingress port traffic which is not intended for them (this is different than shared-bus PCI(X), where devices commonly decode addresses

and commands not targeting them). Endpoint routing is limited to accepting or rejecting transactions presented to them.

System Routing Strategy Is Programmed

Before transactions can be generated by a requester, accepted by the completer, and forwarded by any devices in the path between the two, all devices must be configured to enforce the system transaction routing scheme. Routing is based on traffic type, system memory and IO address assignments, etc. In keeping with PCI plug-and-play configuration methods, each PCI express device is discovered, memory and IO address resources are assigned to them, and switch/bridge devices are programmed to forward transactions on their behalf. Once routing is programmed, bus mastering and target address decoding are enabled. Thereafter, devices are prepared to generate, accept, forward, or reject transactions as necessary.

Two Types of Local Link Traffic

Local traffic occurs between the transmit interface of one device and the receive interface of its neighbor for the purpose of managing the link itself. This traffic is never forwarded or flow controlled; when sent, it must be accepted. Local traffic is further classified as *Ordered Sets* exchanged between the Physical Layers of two devices on a link or Data Link Layer packets (DLLPs) exchanged between the Data Link Layers of the two devices.

Ordered Sets

These are sent by each physical layer transmitter to the physical layer of the corresponding receiver to initiate link training, compensate for clock tolerance, or transition a link to and from the Electrical Idle state. As indicated in Table 3-1 on page 109, there are five types of Ordered Sets.

Each ordered set is constructed of 10-bit control (K) symbols that are created within the physical layer. These symbols have a common name as well as a alph-numeric code that defines the 10 bits pattern of 1s and 0s, of which they are comprised. For example, the SKP (Skip) symbol has a 10-bit value represented as K28.0.

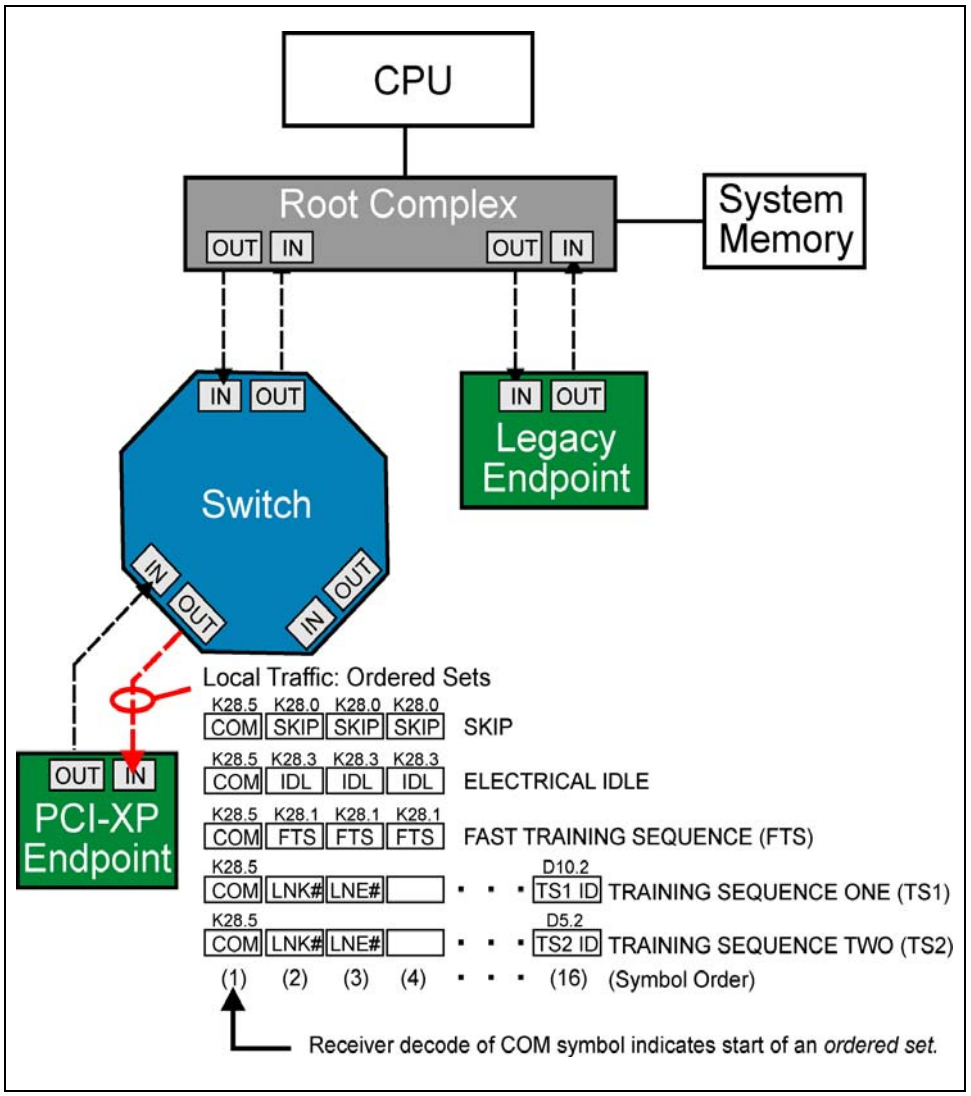
Figure 3-2 on page 110 illustrates the transmission of Ordered Sets. Note that each ordered set is fixed in size, consisting of 4 or 16 characters. Again, the receiver is required to consume them as they are sent. Note that the COM control symbol (K28.5) is used to indicate the start of any ordered set.

Refer to the "8b/10b Encoding" on page 419 for a thorough discussion of Ordered Sets.

Table 3-1: Ordered Set Types

Ordered Set Type	Symbols	Purpose
Fast Training Sequence (FTS)	COM, 3 FTS	Quick synchronization of bit stream when leaving L0s power state.
Training Sequence One (TS1)	COM, Lane ID, 14 more	Used in link training, to align and synchronize the incoming bit stream at startup, convey reset, other functions.
Training Sequence Two (TS2)	COM, Lane ID, 14 more	See TS1.
Electrical Idle (IDLE)	COM, 3 IDL	Indicates that link should be brought to a lower power state (L0s, L1, L2).
Skip	COM, 3 SKP	Inserted periodically to compensate for clock tolerances.

Figure 3-2: PCI Express Link Local Traffic: Ordered Sets



4 Packet-Based Transactions

The Previous Chapter

The previous chapter described the general concepts of PCI Express transaction routing and the mechanisms used by a device in deciding whether to accept, forward, or reject a packet arriving at an ingress port. Because Data Link Layer Packets (DLLPs) and Physical Layer *ordered set* link traffic are never forwarded, the emphasis here is on Transaction Layer Packet (TLP) types and the three routing methods associated with them: address routing, ID routing, and implicit routing. Included is a summary of configuration methods used in PCI Express to set up PCI-compatible plug-and-play addressing within system IO and memory maps, as well as key elements in the PCI Express packet protocol used in making routing decisions.

This Chapter

Information moves between PCI Express devices in packets, and the two major classes of packets are *Transaction Layer Packets* (TLPs), and *Data Link Layer Packets* (DLLPs). The use, format, and definition of all TLP and DLLP packet types and their related fields are detailed in this chapter.

The Next Chapter

The next chapter discusses the Ack/Nak Protocol that verifies the delivery of TLPs between each port as they travel between the requester and completer devices. This chapter details the hardware retry mechanism that is automatically triggered when a TLP transmission error is detected on a given link.

Introduction to the Packet-Based Protocol

The PCI Express protocol improves upon methods used by earlier busses (e.g. PCI) to exchange data and to signal system events. In addition to supporting basic memory, IO, and configuration read/write transactions, the links eliminate many sideband signals and replaces them with in-band messages.

With the exception of the logical idle indication and physical layer *Ordered Sets*, all information moves across an active PCI Express link in fundamental chunks called packets which are comprised of 10 bit control (K) and data (D) symbols. The two major classes of packets exchanged between two PCI Express devices are high level *Transaction Layer Packets* (TLPs), and low-level link maintenance packets called *Data Link Layer Packets* (DLLPs). Collectively, the various TLPs and DLLPs allow two devices to perform memory, IO, and Configuration Space transactions reliably and use messages to initiate power management events, generate interrupts, report errors, etc. Figure 4-1 on page 155 depicts TLPs and DLLPs on a PCI Express link.

Why Use A Packet-Based Transaction Protocol

There are some distinct advantages in using a packet-based protocol, especially when it comes to data integrity. Three important aspects of PCI Express packet protocol help promote data integrity during link transmission:

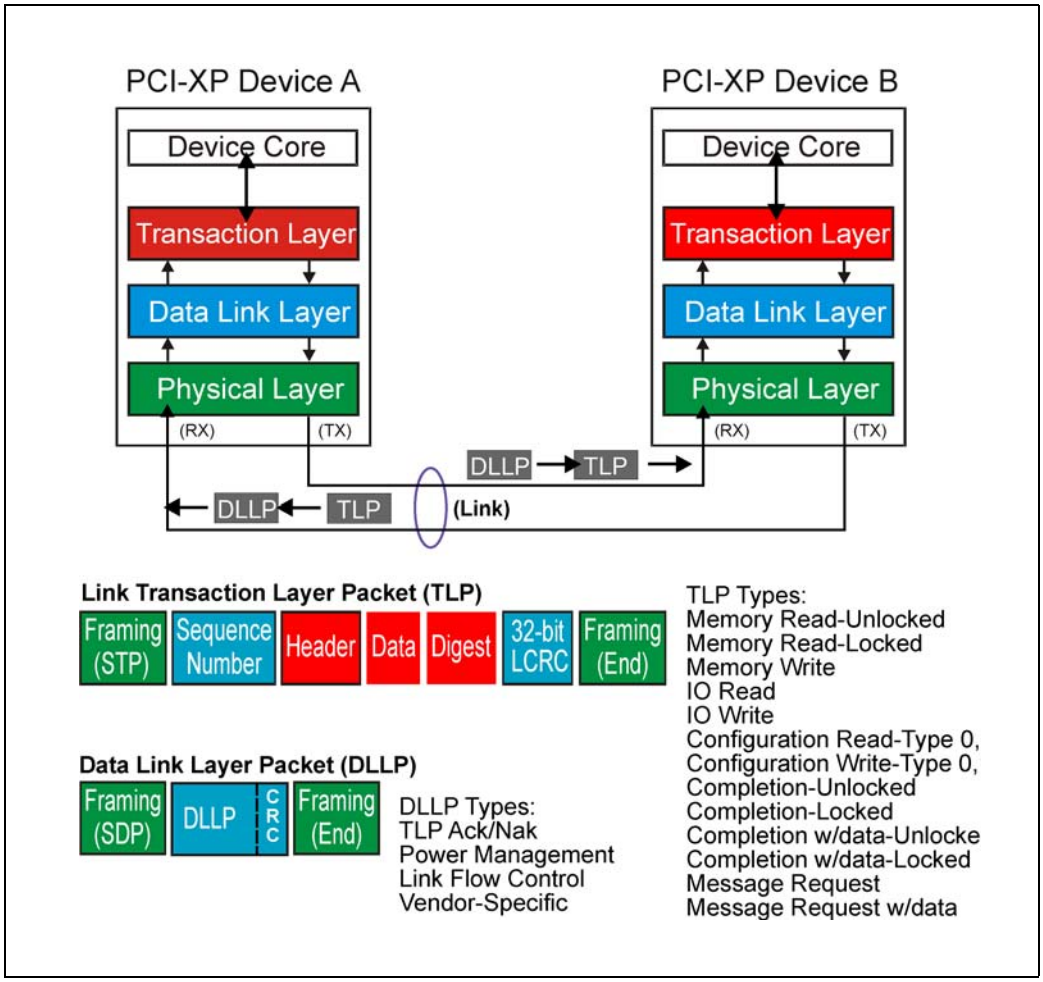
Packet Formats Are Well Defined

Some early bus protocols (e.g. PCI) allow transfers of indeterminate (and unlimited) size, making identification of payload boundaries impossible until the end of the transfer. In addition, an early transaction end might be signaled by either agent (e.g. target disconnect on a write or pre-emption of the initiator during a read), resulting in a partial transfer. In these cases, it is difficult for the sender of data to calculate and send a checksum or CRC covering an entire payload, when it may terminate unexpectedly. Instead, PCI uses a simple parity scheme which is applied and checked for each bus phase completed.

In contrast, each PCI Express packet has a known size and format, and the packet *header*--positioned at the beginning of each DLLP and TLP packet-- indicates the packet type and presence of any optional fields. The size of each packet field is either fixed or defined by the packet type. The size of any data payload is conveyed in the TLP header *Length* field. Once a transfer commences, there are no early transaction terminations by the recipient. This structured

packet format makes it possible to insert additional information into the packet into prescribed locations, including framing symbols, CRC, and a packet sequence number (TLPs only).

Figure 4-1: TLP And DLLP Packets



Framing Symbols Indicate Packet Boundaries

Each TLP and DLLP packet sent is framed with a Start and End control symbol, clearly defining the packet boundaries to the receiver. Note that the Start and End control (K) symbols appended to packets by the transmitting device are 10 bits each. This is a big improvement over PCI and PCI-X which use the assertion and de-assertion of a single FRAME# signal to indicate the beginning and end of a transaction. A glitch on the FRAME# signal (or any of the other PCI/PCIX control signals) could cause a target to misconstrue bus events. In contrast, a PCI Express receiver must properly decode a complete 10 bit symbol before concluding link activity is beginning or ending. Unexpected or unrecognized control symbols are handled as errors.

CRC Protects Entire Packet

Unlike the side-band parity signals used by PCI devices during the address and each data phase of a transaction, the in-band 16-bit or 32-bit PCI Express CRC value "protects" the entire packet (other than framing symbols). In addition to CRC, TLP packets also have a packet *sequence number* appended to them by the transmitter so that if an error is detected at the receiver, the specific packet(s) which were received in error may be resent. The transmitter maintains a copy of each TLP sent in a *Retry Buffer* until it is checked and acknowledged by the receiver. This TLP acknowledgement mechanism (sometimes referred to as the *Ack/Nak* protocol) forms the basis of link-level TLP error correction and is very important in deep topologies where devices may be many links away from the host in the event an error occurs and CPU intervention would otherwise be needed.

Transaction Layer Packets

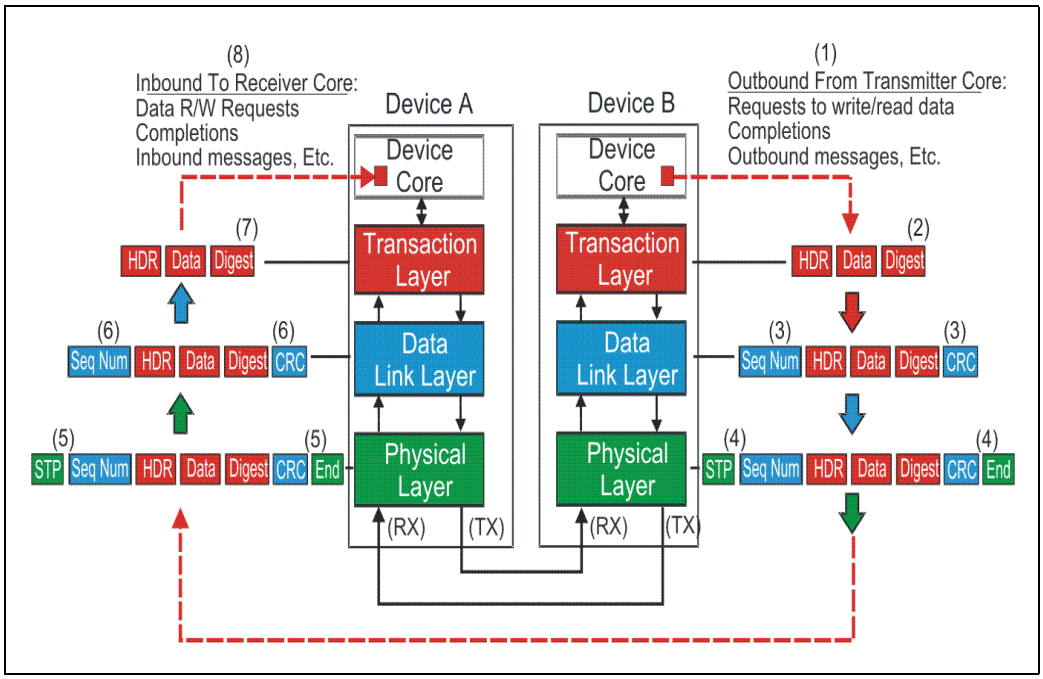
In PCI Express terminology, high-level transactions originate at the device core of the transmitting device and terminate at the core of the receiving device. The Transaction Layer is the starting point in the assembly of outbound Transaction Layer Packets (TLPs), and the end point for disassembly of inbound TLPs at the receiver. Along the way, the Data Link Layer and Physical Layer of each device contribute to the packet assembly and disassembly as described below.

TLPs Are Assembled And Disassembled

Figure 4-2 on page 158 depicts the general flow of TLP assembly at the transmit side of a link and disassembly at the receiver. The key stages in Transaction Layer Packet protocol are listed below. The numbers correspond to those in Figure 4-2.

1. Device B's core passes a request for service to the PCI Express hardware interface. How this done is not covered by the PCI Express Specification, and is device-specific. General information contained in the request would include:
   - The PCI Express command to be performed
   - Start address or ID of target (if address routing or ID routing are used)
   - Transaction type (memory read or write, configuration cycle, etc.)
   - Data payload size (and the data to send, if any)
   - Virtual Channel/Traffic class information
   - Attributes of the transfer: No Snoop bit set?, Relaxed Ordering set?, etc.
2. The Transaction Layer builds the TLP header, data payload, and digest based on the request from the core. Before sending a TLP to the Data Link Layer, flow control credits and ordering rules must be applied.
3. When the TLP is received at the Data Link Layer, a Sequence Number is assigned and a Link CRC is calculated for the TLP (includes Sequence Number). The TLP is then passed on to the Physical Layer.
4. At the Physical Layer, byte striping, scrambling, encoding, and serialization are performed. STP and END control (K) characters are appended to the packet. The packet is sent out on the transmit side of the link.
5. At the Physical Layer receiver of Device A, de-serialization, framing symbol check, decoding, and byte un-striping are performed. Note that at the Physical Layer, the first level or error checking is performed (on the control codes).
6. The Data Link Layer of the receiver calculates CRC and checks it against the received value. It also checks the Sequence Number of the TLP for violations. If there are no errors, it passes the TLP up to the Transaction Layer of the receiver. The information is decoded and passed to the core of Device A. The Data Link Layer of the receiver will also notify the transmitter of the success or failure in processing the TLP by sending an Ack or Nak DLLP to the transmitter. In the event of a Nak (No Acknowledge), the transmitter will re-send all TLPs in its Retry Buffer.

Figure 4-2: PCI Express Layered Protocol And TLP Assembly/Disassembly



5 ACK/NAK Protocol

The Previous Chapter

Information moves between PCI Express devices in packets. The two major classes of packets are Transaction Layer Packets (TLPs), and Data Link Layer Packets (DLLPs). The use, format, and definition of all TLP and DLLP packet types and their related fields were detailed in that chapter.

This Chapter

This chapter describes a key feature of the Data Link Layer: 'reliable' transport of TLPs from one device to another device across the Link. The use of ACK DLLPs to confirm reception of TLPs and the use of NAK DLLPs to indicate error reception of TLPs is explained. The chapter describes the rules for replaying TLPs in the event that a NAK DLLP is received.

The Next Chapter

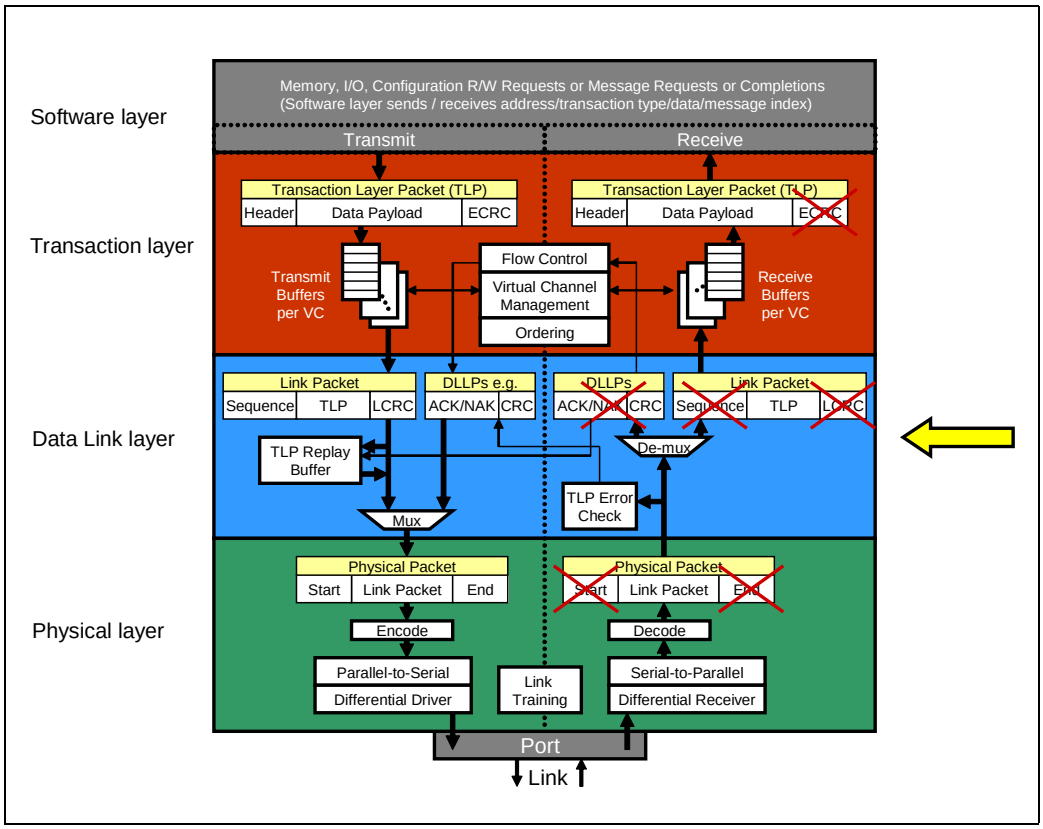
The next chapter discusses Traffic Classes, Virtual Channels, and Arbitration that support Quality of Service concepts in PCI Express implementations. The concept of Quality of Service in the context of PCI Express is an attempt to predict the bandwidth and latency associated with the flow of different transaction streams traversing the PCI Express fabric. The use of QoS is based on application-specific software assigning Traffic Class (TC) values to transactions, which define the priority of each transaction as it travels between the Requester and Completer devices. Each TC is mapped to a Virtual Channel (VC) that is used to manage transaction priority via two arbitration schemes called port and VC arbitration.

Reliable Transport of TLPs Across Each Link

The function of the Data Link Layer (shown in Figure 5-1 on page 210) is two fold:

- 'Reliable' transport of TLPs from one device to another device across the Link.
- The receiver's Transaction Layer should receive TLPs in the same order that the transmitter sent them. The Data Link Layer must preserve this order despite any occurrence of errors that require TLPs to be replayed (retried).

Figure 5-1: Data Link Layer



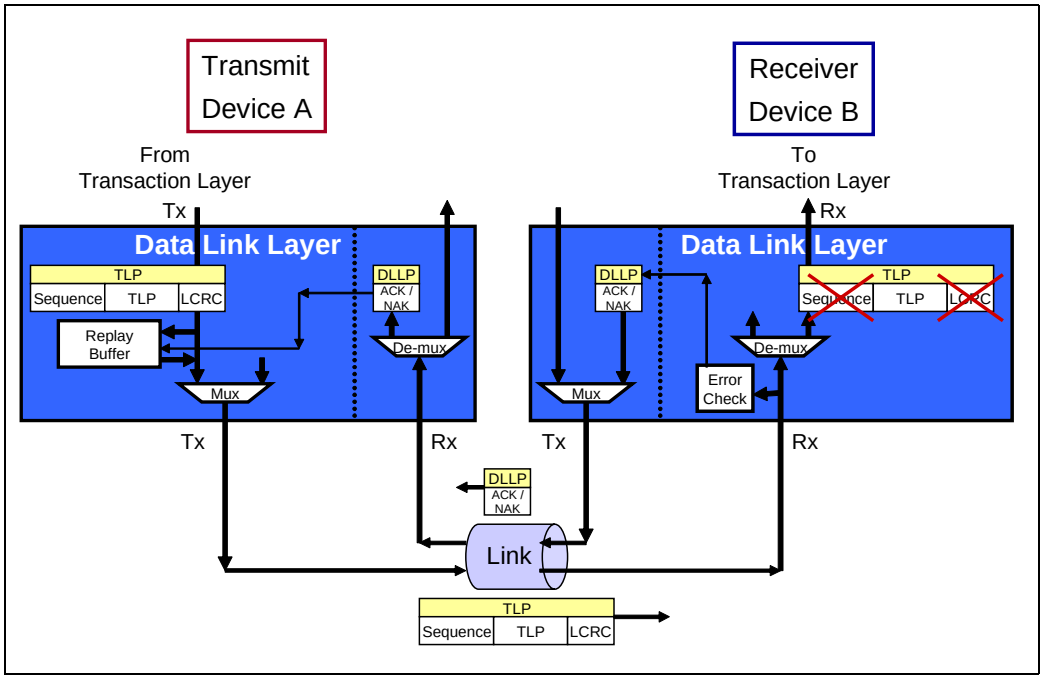
The ACK/NAK protocol associated with the Data Link Layer is described with the aid of Figure 5-2 on page 211 which shows sub-blocks with greater detail. For every TLP that is sent from one device (Device A) to another (Device B) across one Link, the receiver checks for errors in the TLP (using the TLP's LCRC field). The receiver Device B notifies transmitter Device A on good or bad reception of TLPs by returning an ACK or a NAK DLLP. Reception of an ACK DLLP by the transmitter indicates that the receiver has received one or more TLP(s) successfully. Reception of a NAK DLLP by the transmitter indicates that the receiver has received one or more TLP(s) in error. Device A which receives a NAK DLLP then re-sends associated TLP(s) which will hopefully, arrive at the receiver successfully without error.

The error checking capability in the receiver and the transmitter's ability to re-send TLPs if a TLP is not received correctly is the core of the ACK/NAK protocol described in this chapter.

Definition: As used in this chapter, the term Transmitter refers to the device that sends TLPs.

Definition: As used in this chapter, the term Receiver refers to the device that receives TLPs.

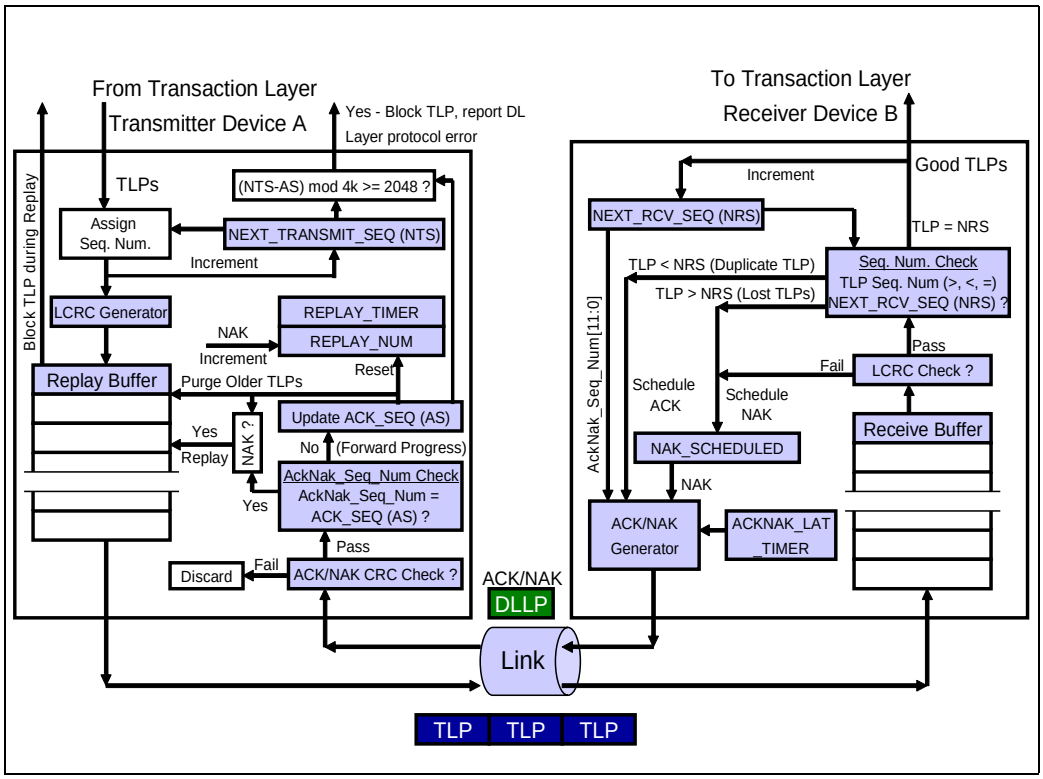
Figure 5-2: Overview of the ACK/NAK Protocol



Elements of the ACK/NAK Protocol

Figure 5-3 is a block diagram of a transmitter and a remote receiver connected via a Link. The diagram shows all of the major Data Link Layer elements associated with reliable TLP transfer from the transmitter's Transaction Layer to the receiver's Transaction Layer. Packet order is maintained by the transmitter's and receiver's Transaction Layer.

Figure 5-3: Elements of the ACK/NAK Protocol



Transmitter Elements of the ACK/NAK Protocol

Figure 5-4 on page 215 illustrates the transmitter Data Link Layer elements associated with processing of outbound TLPs and inbound ACK/NAK DLLPs.

Replay Buffer

The replay buffer stores TLPs with all fields including the Data Link Layer-related Sequence Number and LCRC fields. The TLPs are saved in the order of arrival from the Transaction Layer before transmission. Each TLP in the Replay Buffer contains a Sequence Number which is incrementally greater than the sequence number of the previous TLP in the buffer.

When the transmitter receives acknowledgement via an ACK DLLP that TLPs have reached the receiver successfully, it purges the associated TLPs from the Replay Buffer. If, on the other hand, the transmitter receives a NAK DLLP, it replays (i.e., re-transmits) the contents of the buffer.

NEXT_TRANSMIT_SEQ Counter

This counter generates the Sequence Number assigned to each new transmitted TLP. The counter is a 12-bit counter that is initialized to 0 at reset, or when the Data Link Layer is in the inactive state. It increments until it reaches 4095 and then rolls over to 0 (i.e., it is a modulo 4096 counter).

LCRC Generator

The LCRC Generator provides a 32-bit LCRC for the TLP. The LCRC is calculated using all fields of the TLP including the Header, Data Payload, ECRC and Sequence Number. The receiver uses the TLP's LCRC field to check for a CRC error in the received TLP.

REPLAY_NUM Count

This 2-bit counter stores the number of replay attempts following either reception of a NAK DLLP, or a REPLAY_TIMER time-out. When the REPLAY_NUM count rolls over from 11b to 00b, the Data Link Layer triggers a Physical Layer Link-retrain (see the description of the LTSSM recovery state on page 532). It waits for completion of re-training before attempting to transmit TLPs once again. The REPLAY_NUM counter is initialized to 00b at reset, or when the Data Link Layer is inactive. It is also reset whenever an ACK is received, indicating that forward progress is being made in transmitting TLPs.

REPLAY_TIMER Count

The REPLAY_TIMER is used to measure the time from when a TLP is transmitted until an associated ACK or NAK DLLP is received. The REPLAY_TIMER is started (or restarted, if already running) when the last Symbol of any TLP is sent. It restarts from 0 each time that there are outstanding TLPs in the Replay Buffer and an ACK DLLP is received that references a TLP still in the Replay Buffer. It resets to 0 and holds when there are no outstanding TLPs in the Replay Buffer, or until restart conditions are met for each NAK received (except during a replay), or when the REPLAY_TIMER expires. It is not advanced (i.e., its value remains fixed) during Link re-training.

ACKD_SEQ Count

This 12-bit register tracks or stores the Sequence Number of the most recently received ACK or NAK DLLP. It is initialized to all 1s at reset, or when the Data Link Layer is inactive. This register is updated with the AckNak_Seq_Num [11:0] field of a received ACK or NAK DLLP. The ACKD_SEQ count is compared with the NEXT_TRANSMIT_SEQ count.

IF (NEXT_TRANSMIT_SEQ - ACKD_SEQ) mod 4096 ≥ 2048 THEN
New TLPs from Transaction Layer are not accepted by Data Link Layer until this equation is no longer true. In addition, a Data Link Layer protocol error which is a fatal uncorrectable error is reported. This error condition occurs if there is a separation greater than 2047 between NEXT_TRANSMIT_SEQ and ACKD_SEQ. i.e, a separation greater than 2047 between the sequence number of a TLP being transmitted and that of a TLP in the replay buffer that receives an ACK or NAK DLLP.

Also, the ACKD_SEQ count is used to check for forward progress made in transmitting TLPs. If no forward progress is made after 3 additional replay attempts, the Link in re-trained.

DLLP CRC Check

This block checks for CRC errors in DLLPs returned from the receiver. Good DLLPs are further processed. If a DLLP CRC error is detected, the DLLP is discarded and an error reported. No further action is taken.

Definition: The Data Link Layer is in the inactive state when the Physical Layer reports that the Link is non-operational or nothing is connected to the Port. The Physical Layer is in the non-operational state when the Link Training and Status State Machine (LTSSM) is in the Detect, Polling, Configuration, Disabled, Reset

6 QoS/TCs/VCs and Arbitration

The Previous Chapter

The previous chapter detailed the Ack/Nak Protocol that verifies the delivery of TLPs between each port as they travel between the requester and completer devices. This chapter details the hardware retry mechanism that is automatically triggered when a TLP transmission error is detected on a given link.

This Chapter

This chapter discusses Traffic Classes, Virtual Channels, and Arbitration that support Quality of Service concepts in PCI Express implementations. The concept of Quality of Service in the context of PCI Express is an attempt to predict the bandwidth and latency associated with the flow of different transaction streams traversing the PCI Express fabric. The use of QoS is based on application-specific software assigning Traffic Class (TC) values to transactions, which define the priority of each transaction as it travels between the Requester and Completer devices. Each TC is mapped to a Virtual Channel (VC) that is used to manage transaction priority via two arbitration schemes called port and VC arbitration.

The Next Chapter

The next chapter discusses the purposes and detailed operation of the Flow Control Protocol. This protocol requires each device to implement credit-based link flow control for each virtual channel on each port. Flow control guarantees that transmitters will never send Transaction Layer Packets (TLPs) that the receiver can't accept. This prevents receive buffer over-runs and eliminates the need for inefficient disconnects, retries, and wait-states on the link. Flow Control also helps enable compliance with PCI Express ordering rules by maintaining separate virtual channel Flow Control buffers for three types of transactions: Posted (P), Non-Posted (NP) and Completions (Cpl).

Quality of Service

Quality of Service (QoS) is a generic term that normally refers to the ability of a network or other entity (in our case, PCI Express) to provide predictable latency and bandwidth. QoS is of particular interest when applications require guaranteed bus bandwidth at regular intervals, such as audio data. To help deal with this type of requirement PCI Express defines isochronous transactions that require a high degree of QoS. However, QoS can apply to any transaction or series of transactions that must traverse the PCI Express fabric. Note that QoS can only be supported when the system and device-specific software is PCI Express aware.

QoS can involve many elements of performance including:

- Transmission rate
- Effective Bandwidth
- Latency
- Error rate
- Other parameters that affect performance

Several features of PCI Express architecture provide the mechanisms that make QoS achievable. The PCI Express features that support QoS include:

- Traffic Classes (TCs)
- Virtual Channels (VCs)
- Port Arbitration
- Virtual Channel Arbitration
- Link Flow Control

PCI Express uses these features to support two general classes of transactions that can benefit from the PCI Express implementation of QoS.

Isochronous Transactions — from Iso (same) + chronous (time), these transactions require a constant bus bandwidth at regular intervals along with guaranteed latency. Isochronous transactions are most often used when a synchronous connection is required between two devices. For example, a CD-ROM drive containing a music CD may be sourcing data to speakers. A synchronous connection exists when a headset is plugged directly into the drive. However, when the audio card is used to deliver the audio information to a set of external speakers, isochronous transactions may be used to simplify the delivery of the data.

Asynchronous Transactions — This class of transactions involves a wide variety of applications that have widely varying requirements for bandwidth and latency. QoS can provide the more demanding applications (those requiring higher bandwidth and shorter latencies) with higher priority than the less demanding applications. In this way, software can establish a hierarchy of traffic classes for transactions that permits differentiation of transaction priority based on their requirements. The specification refers to this capability as differentiated services.

Isochronous Transaction Support

PCI Express supports QoS and the associated TC, VC, and arbitration mechanisms so that isochronous transactions can be performed. A classic example of a device that benefits from isochronous transaction support is a video camera attached to a tape deck. This real-time application requires that image and audio data be transferred at a constant rate (e.g., 64 frames/second). This type of application is typically supported via a direct synchronous attachment between the two devices.

Synchronous Versus Isochronous Transactions

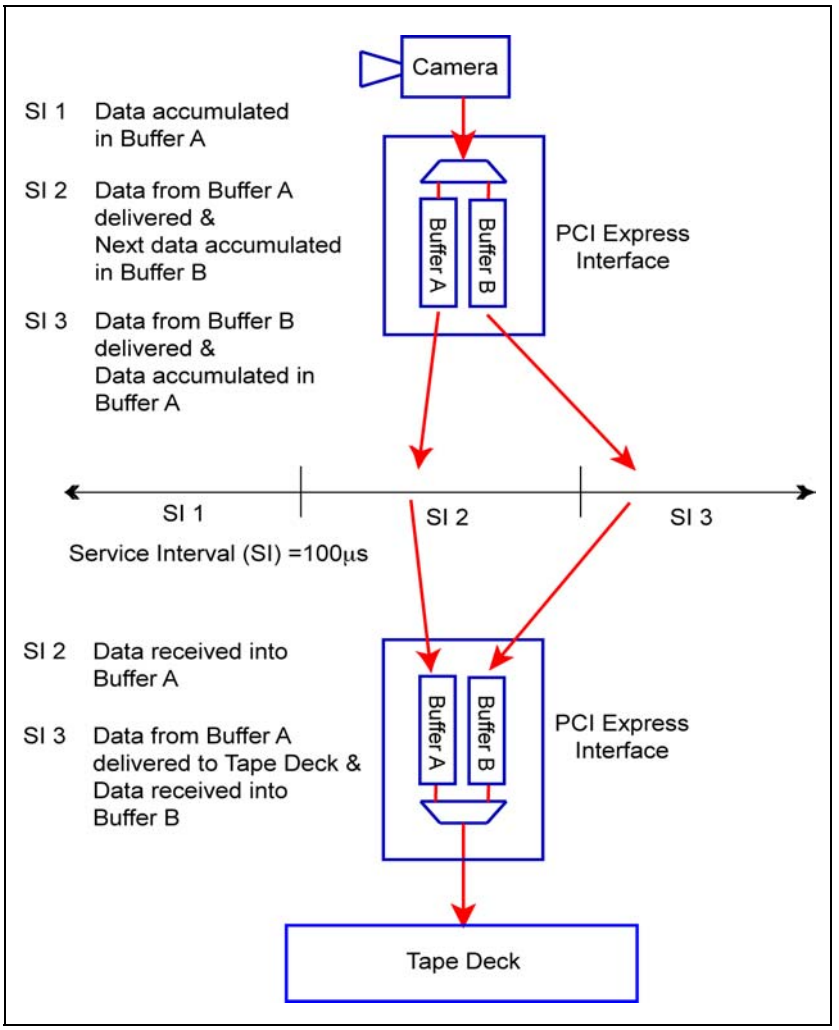
Two devices connected directly perform synchronous transfers. A synchronous source delivers data directly to the synchronous sink through use of a common reference clock. In our example, the video camera (synchronous source) sends audio and video data to the tape deck (synchronous sink), which immediately stores the data in real time with little or no data buffering, and with only a slight delay due to signal propagation.

When these devices are connected via PCI Express a synchronous connection is not possible. Instead, PCI Express emulates synchronous connections through the use of isochronous transactions and data buffering. In this scenario, isochronous transactions can be used to ensure that a constant amount of data is delivered at specified intervals (100µs in this example), thus achieving the required transmission characteristics. Consider the following sequence (Refer to Figure 6-1 on page 254):

1. The synchronous source (video camera and PCI Express interface) accumulates data in Buffer A during service interval 1 (SI 1).
2. The camera delivers the accumulated data to the synchronous sink (tape deck) sometime during the next service interval (SI 2). The camera also accumulates the next block of data in Buffer B as the contents of Buffer A is delivered.

3. The tape deck buffers the incoming data (in its Buffer A), which can then be delivered synchronously for recording on tape during service interval 3. During SI 3 the camera once again accumulates data into Buffer A, and the cycle repeats.

Figure 6-1: Example Application of Isochronous Transaction



Isochronous Transaction Management

Management of an isochronous communications channel is based on a Traffic Class (TC) value and an associated Virtual Channel (VC) number that software assigns during initialization. Hardware components including the Requester of a transaction and all devices in the path between the requester and completer are configured to transport the isochronous transactions from link to link via a hi-priority virtual channel.

The requester initiates isochronous transactions that include a TC value representing the desired QoS. The Requester injects isochronous packets into the fabric at the required rate (service interval), and all devices in the path between the Requester and Completer must be configured to support the transport of the isochronous transactions at the specified interval. Any intermediate device along the path must convert the TC to the associated VC used to control transaction arbitration. This arbitration results in the desired bandwidth and latency for transactions with the assigned TC. Note that the TC value remains constant for a given transaction while the VC number may change from link to link.

Differentiated Services

Various types of asynchronous traffic (all traffic other than isochronous) have different priority from the system perspective. For example, ethernet traffic requires higher priority (smaller latencies) than mass storage transactions. PCI Express software can establish different TC values and associated virtual channels and can set up the communications paths to ensure different delivery policies are established as required. Note that the specification does not define specific methods for identifying delivery requirements or the policies to be used when setting up differentiated services.

Perspective on QOS/TC/VC and Arbitration

PCI does not include any QoS-related features similar to those defined by PCI Express. Many questions arise regarding the need for such an elaborate scheme for managing traffic flow based on QoS and differentiated services. Without implementing these new features, the bandwidth available with a PCI Express system is far greater and latencies much shorter than PCI-based implementations, due primarily to the topology and higher delivery rates. Consequently, aside from the possible advantage of isochronous transactions, there appears to be little advantage to implementing systems that support multiple Traffic

Classes and Virtual Channels.

While this may be true for most desktop PCs, other high-end applications may benefit significantly from these new features. The PCI Express specification also opens the door to applications that demand the ability to differentiate and manage system traffic based on Traffic Class prioritization.

Traffic Classes and Virtual Channels

During initialization a PCI Express device-driver communicates the levels of QoS that it desires for its transactions, and the operating system returns TC values that correspond to the QoS requested. The TC value ultimately determines the relative priority of a given transaction as it traverses the PCI Express fabric. Two hardware mechanisms provide guaranteed isochronous bandwidth and differentiated services:

- Virtual Channel Arbitration
- Port Arbitration

These arbitration mechanisms use VC numbers to manage transaction priority. System configuration software must assign VC IDs and set up the association between the traffic class assigned to a transaction and the virtual channel to be used when traversing each link. This is done via VC configuration registers mapped within the extended configuration address space. The list of these registers and their location within configuration space is illustrated in Figure 6-2.

7 Flow Control

The Previous Chapter

This previous chapter discussed Traffic Classes, Virtual Channels, and Arbitration that supports Quality of Service concepts in PCI Express implementations. The concept of Quality of Service in the context of PCI Express is an attempt to predict the bandwidth and latency associated with the flow of different transaction streams traversing the PCI Express fabric. The use of QoS is based on application-specific software assigning Traffic Class (TC) values to transactions, which define the priority of each transaction as it travels between the Requester and Completer devices. Each TC is mapped to a Virtual Channel (VC) that is used to manage transaction priority via two arbitration schemes called port and VC arbitration.

This Chapter

This chapter discusses the purposes and detailed operation of the Flow Control Protocol. This protocol requires each device to implement credit-based link flow control for each virtual channel on each port. Flow control guarantees that transmitters will never send Transaction Layer Packets (TLPs) that the receiver can't accept. This prevents receive buffer over-runs and eliminates the need for inefficient disconnects, retries, and wait-states on the link. Flow Control also helps enable compliance with PCI Express ordering rules by maintaining separate virtual channel Flow Control buffers for three types of transactions: Posted (P), Non-Posted (NP) and Completions (Cpl).

The Next Chapter

The next chapter discusses the ordering requirements for PCI Express devices, as well as PCI and PCI-X devices that may be attached to a PCI Express fabric. The discussion describes the Producer/Consumer programming model upon which the fundamental ordering rules are based. It also describes the potential performance problems that can emerge when strong ordering is employed, describes the weak ordering solution, and specifies the rules defined for deadlock avoidance.

Flow Control Concept

The ports at each end of every PCI Express link must implement Flow Control. Before a transaction packet can be sent across a link to the receiving port, the transmitting port must verify that the receiving port has sufficient buffer space to accept the transaction to be sent. In many other architectures including PCI and PCI-X, transactions are delivered to a target device without knowing if it can accept the transaction. If the transaction is rejected due to insufficient buffer space, the transaction is resent (retried) until the transaction completes. This procedure can severely reduce the efficiency of a bus, by wasting bus bandwidth when other transactions are ready to be sent.

Because PCI Express is a point-to-point implementation, the Flow Control mechanism would be ineffective, if only one transaction stream was pending transmission across a link. That is, if the receive buffer was temporarily full, the transmitter would be prevented from sending a subsequent transaction due to transaction ordering requirements, thereby blocking any further transfers. PCI Express improves link efficiency by implementing multiple flow-control buffers for separate transaction streams (virtual channels). Because Flow Control is managed separately for each virtual channel implemented for a given link, if the Flow Control buffer for one VC is full, the transmitter can advance to another VC buffer and send transactions associated with it.

The link Flow Control mechanism uses a credit-based mechanism that allows the transmitting port to check buffer space availability at the receiving port. During initialization each receiver reports the size of its receive buffers (in Flow Control credits) to the port at the opposite end of the link. The receiving port continues to update the transmitting port regularly by transmitting the number of credits that have been freed up. This is accomplished via Flow Control DLLPs.

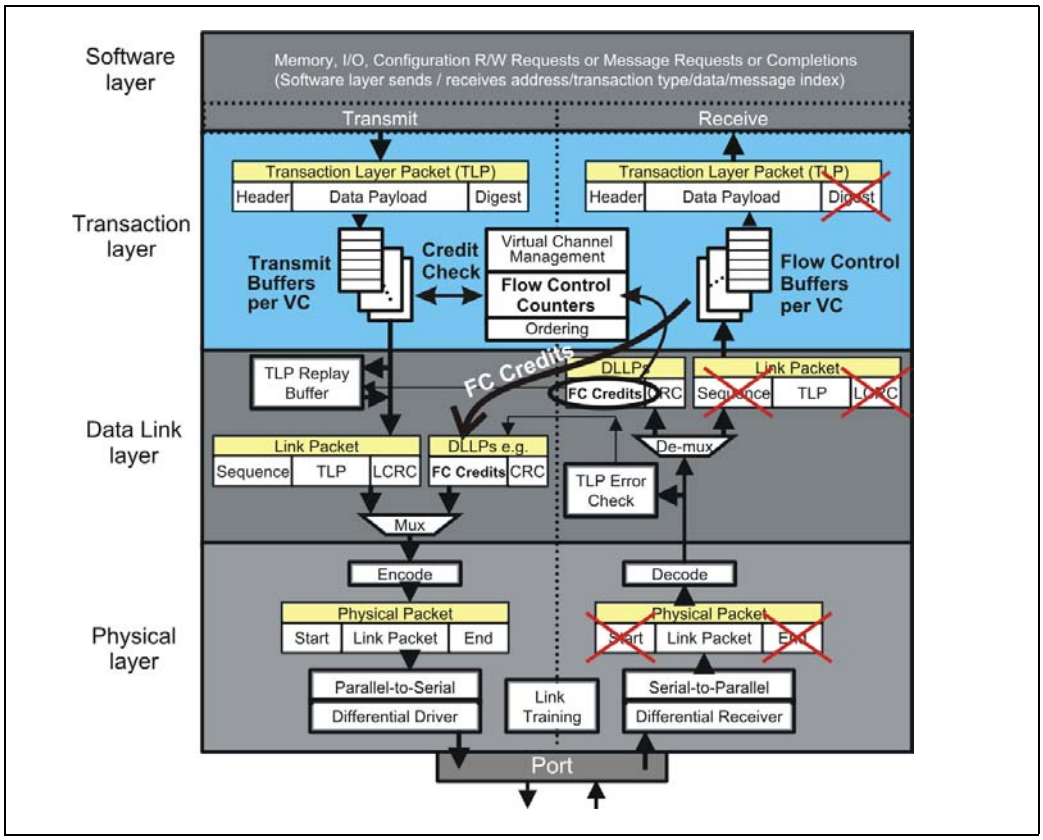
Flow control logic is located in the transaction layer of the transmitting and receiving devices. Both transmitter and receiver sides of each device are involved in flow control. Refer to Figure 7-1 on page 287 during the following descriptions.

- **Devices Report Buffer Space Available** — The receiver of each node contains the Flow Control buffers. Each device must report the amount of flow control buffer space they have available to the device on the opposite end of the link. Buffer space is reported in units called Flow Control Credits (FCCs). The number of Flow Control Credits within each buffer is forwarded from the transaction layer to the transmit side of the link layer as

illustrated in Figure 7-1. The link creates a Flow Control DLLP that carries this credit information to the receiver at the opposite end of the link. This is done for each Flow Control Buffer.

- **Receiving Credits** — Notice that the receiver in Figure 7-1 also receives Flow Control DLLPs from the device at the opposite end of the link. This information is transferred to the transaction layer to update the Flow Control Counters that track the amount of Flow Control Buffer space in the other device.
- **Credit Checks Made** — Each transmitter check consults the Flow Control Counters to check available credits. If sufficient credits are available to receive the transaction pending delivery then the transaction is forwarded to the link layer and is ultimately sent to the opposite device. If enough credits are not available the transaction is temporarily blocked until additional Flow Control credits are reported by the receiving device.

Figure 7-1: Location of Flow Control Logic



Flow Control Buffers

Flow control buffers are implemented for each VC resource supported by a PCI Express port. Recall that devices at each end of the link may not support the same number of VC resources, therefore the maximum number of VCs configured and enabled by software is the greatest number of VCs in common between the two ports.

VC Flow Control Buffer Organization

Each VC Flow Control buffer at the receiver is managed for each category of transaction flowing through the virtual channel. These categories are:

- Posted Transactions — Memory Writes and Messages
- Non-Posted Transactions — Memory Reads, Configuration Reads and Writes, and I/O Reads and Writes
- Completions — Read Completions and Write Completions

In addition, each of these categories is separated into header and data portions of each transaction. Flow control operates independently for each of the six buffers listed below (also see Figure 7-2 on page 289).

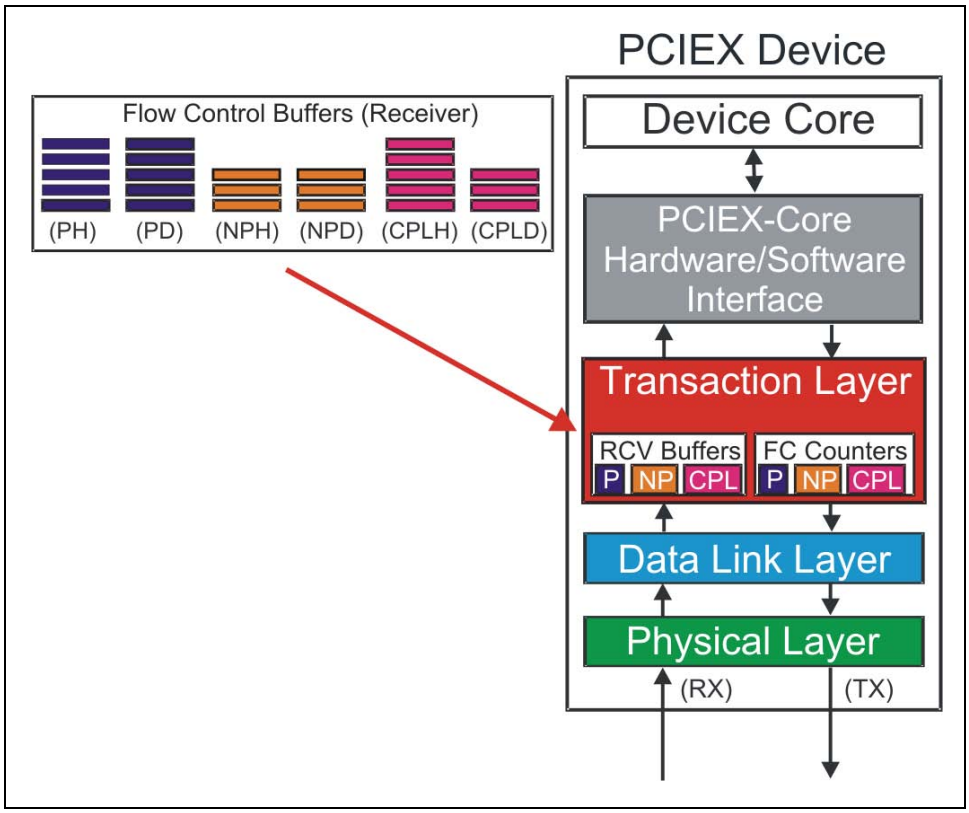
- Posted Header
- Posted Data

- Non-Posted Header
- Non-Posted Data

- Completion Header
- Completion Data

Some transactions consist of a header only (e.g., read requests) while others consist of a header and data (e.g., write requests). The transmitter must ensure that both header and data buffer space is available as required for each transaction before the transaction can be sent. Note that when a transaction is received into a VC Flow Control buffer that ordering must be maintained when the transactions are forwarded to software or to an egress port in the case of a switch. The the receiver must also track the order of header and data components within the Flow Control buffer.

Figure 7-2: Flow Control Buffer Organization



Flow Control Credits

Buffer space is reported by the receiver in units called Flow Control credits. The unit value of Flow Control credits (FCCs) may differ between header and data as listed below:

- Header FCCs — maximum header size + digest
  - o 4 DWs for completions
  - o 5 DWs for requests
- Data FCCs — 4 DWs (aligned 16 bytes)

Flow control credits are passed within the header of the link layer Flow Control Packets. Note that DLLPs do not require Flow Control credits because they originate and terminate at the link layer.

Maximum Flow Control Buffer Size

The maximum buffer size that can be reported via the Flow Control Initialization and Update packets for the header and data portions of a transaction are as follows:

128 Credits for headers

- 2,560 bytes Request Headers @ 20 bytes/credit
- 2048 bytes for completion headers @ 16 bytes/credit

2048 Credits for data

- 32KB @ 16 bytes/credit

The reason for these limits is discussed in the section entitled "Stage 1 — Flow Control Following Initialization" page 296, step 2.

Introduction to the Flow Control Mechanism

The specification defines the requirements of the Flow Control mechanism by describing conceptual registers and counters along with procedures and mechanisms for reporting, tracking, and calculating whether a transaction can be sent. These elements define the functional requirements; however, the actual implementation may vary from the conceptual model. This section introduces the specified model that serves to explain the concept and define the requirements. The approach taken focuses on a single flow control example for a non-posted header. The concepts discussed apply to all Flow Control buffer types.

The Flow Control Elements

Figure 7-3 identifies and illustrates the elements used by the transmitter and receiver when managing flow control. This diagram illustrates transactions flowing in a single direction across a link, but of course another set of these elements is used to support transfers in the opposite direction. The primary function of each element within the transmitting and receiving devices is listed below. Note that for a single direction these Flow Control elements are duplicated for each Flow Control receive buffer, yielding six sets of elements. This example deals with non-posted header flow control.

8 Transaction Ordering

The Previous Chapter

The previous chapter discussed the purposes and detailed operation of the Flow Control Protocol. This protocol requires each device to implement credit-based link flow control for each virtual channel on each port. Flow control guarantees that transmitters will never send Transaction Layer Packets (TLPs) that the receiver can't accept. This prevents receive buffer over-runs and eliminates the need for inefficient disconnects, retries, and wait-states on the link. Flow Control also helps enable compliance with PCI Express ordering rules by maintaining separate Virtual Channel Flow Control buffers for three types of transactions: Posted (P), Non-Posted (NP) and Completions (Cpl).

This Chapter

This chapter discusses the ordering requirements for PCI Express devices as well as PCI and PCI-X devices that may be attached to a PCI Express fabric. The discussion describes the Producer/Consumer programming model upon which the fundamental ordering rules are based. It also describes the potential performance problems that can emerge when strong ordering is employed and specifies the rules defined for deadlock avoidance.

The Next Chapter

Native PCI Express devices that require interrupt support must use the Message Signaled Interrupt (MSI) mechanism defined originally in the PCI 2.2 specification. The next chapter details the MSI mechanism and also describes the legacy support that permits virtualization of the PCI INTx signals required by devices such as PCI Express-to-PCI Bridges.

Introduction

As with other protocols, PCI Express imposes ordering rules on transactions moving through the fabric at the same time. The reasons for the ordering rules include:

- Ensuring that the completion of transactions is deterministic and in the sequence intended by the programmer.
- Avoiding deadlocks conditions.
- Maintaining compatibility with ordering already used on legacy buses (e.g., PCI, PCI-X, and AGP).
- Maximize performance and throughput by minimizing read latencies and managing read/write ordering.

PCI Express ordering is based on the same Producer/Consumer model as PCI. The split transaction protocol and related ordering rules are fairly straight forward when restricting the discussion to transactions involving only native PCI Express devices. However, ordering becomes more complex when including support for the legacy buses mentioned in bullet three above.

Rather than presenting the ordering rules defined by the specification and attempting to explain the rationale for each rule, this chapter takes the building block approach. Each major ordering concern is introduced one at a time. The discussion begins with the most conservative (and safest) approach to ordering, progresses to a more aggressive approach (to improve performance), and culminates with the ordering rules presented in the specification. The discussion is segmented into the following sections:

1. The Producer/Consumer programming model upon which the fundamental ordering rules are based.
2. The fundamental PCI Express device ordering requirements that ensure the Producer/Consumer model functions correctly.
3. The Relaxed Ordering feature that permits violation of the Producer/Consumer ordering when the device issuing a request knows that the transaction is not part of a Producer/Consumer programming sequence.
4. Modification of the strong ordering rules to improve performance.
5. Avoiding deadlock conditions and support for PCI legacy implementations.

Producer/Consumer Model

Readers familiar with the Producer/Consumer programming model may choose to skip this section and proceed directly to "Native PCI Express Ordering Rules" on page 318.

The Producer/Consumer model is a common methodology that two requester-capable devices might use to communicate with each other. Consider the following example scenario:

1. A network adapter begins to receive a stream of compressed video data over the network and performs a series of memory write transactions to deliver the stream of compressed video data into a **Data buffer** in memory (in other words the network adapter is the **Producer** of the data).
2. After the **Producer** moves the data to memory, it performs a memory write transaction to set an indicator (or **Flag**) in a memory location (or a register) to indicate that the data is ready for processing.
3. Another requester (referred to as the **Consumer**) periodically performs a memory read from the **Flag** location to see if there's any data to be processed. In this example, this requester is a video decompressor that will decompress and display the data.
4. When it sees that the **Flag** has been set by the **Producer**, it performs a memory write to clear the **Flag,** followed by a burst memory read transaction to read the compressed data (it consumes the data; hence the name **Consumer**) from the **Data buffer** in memory.
5. When it is done consuming the **Data**, the **Consumer** writes the completion status into the **Status** location. It then resumes periodically reading the **Flag** location to determine when more data needs to be processed.
6. In the meantime, the **Producer** has been reading periodically from the **Status** location to see if data processing has been completed by the other requester (the **Consumer**). This location typically contains zero until the other requester completes the data processing and writes the completion status into it. When the **Producer** reads the **Status** and sees that the **Consumer** has completed processing the **Data**, the **Producer** then performs a memory write to clear the **Status** location.
7. The process then repeats whenever the **Producer** has more data to be processed.

Ordering rules are required to ensure that the Producer/Consumer model works correctly no matter where the **Producer**, the **Consumer**, the **Data** buffer, the **Flag** location, and the **Status** location are located in the system (in other words, no matter how they are distributed on various links in the system).

Native PCI Express Ordering Rules

PCI Express transaction ordering for native devices can be summarized with four simple rules:

1. PCI Express requires strong ordering of transactions (i.e., performing transactions in the order issued by software) flowing through the fabric that have the same TC assignment (see item 4 for the exception to this rule). Because all transactions that have the same TC value assigned to them are mapped to a given VC, the same rules apply to transactions within each VC.
2. No ordering relationship exists between transactions with different TC assignments.
3. The ordering rules apply in the same way to all types of transactions: memory, IO, configuration, and messages.
4. Under limited circumstances, transactions with the Relaxed Ordering attribute bit set can be ordered ahead of other transactions with the same TC.

These fundamental rules ensure that transactions always complete in the order intended by software. However, these rules are extremely conservative and do not necessarily result in optimum performance. For example, when transactions from many devices merge within switches, there may be no ordering relationship between transactions from these different devices. In such cases, more aggressive rules can be applied to improve performance as discussed in "Modified Ordering Rules Improve Performance" on page 322.

Producer/Consumer Model with Native Devices

Because the Producer/Consumer model depends on strong ordering, when the following conditions are met native PCI Express devices support this model without additional ordering rules:

1. All elements associated with the Producer/Consumer model reside within native PCI Express devices.
2. All transactions associated with the operation of the Producer/Consumer model transverse only PCI Express links within the same fabric.
3. All associated transactions have the same TC values. If different TC values are used, then the strong ordering relationship between the transactions is no longer guaranteed.
4. The Relaxed Ordering (RO) attribute bit of the transactions must be cleared to avoid reordering the transactions that are part of the Producer/Consumer transaction series.

When PCI legacy devices reside within a PCI Express system, the ordering rules become more involved. Consequently, additional ordering rules apply because of PCI's delayed transaction protocol. Without ordering rules, this protocol could permit Producer/Consumer transactions to complete out of order and cause the programming model to break.

Relaxed Ordering

PCI Express supports the Relaxed Ordering mechanism introduced by PCI-X; however, PCI Express introduces some changes (discussed later in this chapter). The concept of Relaxed Ordering in the PCI Express environment allows switches in the path between the Requester and Completer to reorder some transactions just received before others that were previously enqueued.

The ordering rules that exist to support the Producer/Consumer model may result in transactions being blocked, when in fact the blocked transactions are completely unrelated to any Producer/Consumer transaction sequence. Consequently, in certain circumstances, a transaction with its Relaxed Ordering (RO) attribute bit set can be re-ordered ahead of other transactions.

The Relaxed Ordering bit may be set by the device if its device driver has enabled it to do so (by setting the Enable Relaxed Ordering bit in the Device Control register—see Table 24 - 3 on page 906). Relaxed ordering gives switches and the Root Complex permission to move this transaction ahead of others, whereas the action is normally prohibited.

RO Effects on Memory Writes and Messages

PCI Express Switches and the Root Complex are affected by memory write and message transactions that have their RO bit set. Memory write and Message transactions are treated the same in most respects—both are handled as posted operations, both are received into the same Posted buffer, and both are subject to the same ordering requirements. When the RO bit is set, switches handle these transactions as follows:

- Switches are permitted to reorder memory write transactions just posted ahead of previously posted memory write transactions or message transactions. Similarly, message transactions just posted may be ordered ahead of previously posted memory write or message transactions. Switches must also forward the RO bit unmodified. The ability to reorder these transactions within switches is not supported by PCI-X bridges. In PCI-X, all

posted writes must be forwarded in the exact order received. Another difference between the PCI-X and PCI Express implementations is that message transactions are not defined for PCI-X.

- The Root Complex is permitted to order a just-posted write transaction ahead of another write transaction that was received earlier in time. Also, when receiving write requests (with RO set), the Root Complex is required to write the data payload to the specified address location within system memory, but is permitted to write each byte to memory in any address order.

RO Effects on Memory Read Transactions

All read transactions in PCI Express are handled as split transactions. When a device issues a memory read request with the RO bit set, the request may traverse one or more switches on its journey to the Completer. The Completer returns the requested read data in a series of one or more split completion transactions, and uses the same RO setting as in the request. Switch behavior for the example stated above is as follow:

1. A switch that receives a memory read request with the RO bit set must forward the request in the order received, and must not reorder it ahead of memory write transactions that were previously posted. This action guarantees that all write transactions moving in the direction of the read request are pushed ahead of the read. Such actions are not necessarily part of the Producer/Consumer programming sequence, but software may depend on this flushing action taking place. Also, the RO bit must not be modified by the switch.
2. When the Completer receives the memory read request, it fetches the requested read data and delivers a series of one or more memory read Completion transactions with the RO bit set (because it was set in the request).
3. A switch receiving the memory read Completion(s) detects the RO bit set and knows that it is allowed to order the read Completion(s) ahead of previously posted memory writes moving in the direction of the Completion. If the memory write transaction were blocked (due to flow control), then the memory read Completion would also be blocked if the RO was not set. Relaxed ordering in this case improves read performance.

Table 8-1 summarizes the relaxed ordering behavior allowed by switches.

9 Interrupts

The Previous Chapter

This chapter discusses the ordering requirements for PCI Express devices as well as PCI and PCI-X devices that may be attached to a PCI Express fabric. The discussion describes the Producer/Consumer programming model upon which the fundamental ordering rules are based. It also describes the potential performance problems that can emerge when strong ordering is employed and specifies the rules defined for deadlock avoidance.

This Chapter

Native PCI Express devices that require interrupt support must use the Message Signaled Interrupt (MSI) mechanism defined originally in the PCI 2.2 version of the specification. This chapter details the MSI mechanism and also describes the legacy support that permits virtualization of the PCI INTx signals required by devices such as PCI Express-to-PCI Bridges.

The Next Chapter

To this point it has been presumed that transactions traversing the fabric have not encountered any errors that cannot be corrected by hardware. The next chapter discusses both correctable and non-correctable errors and discusses the mechanisms used to report them. The PCI Express architecture provides a rich set of error detection, reporting, and logging capabilities. PCI Express error reporting classifies errors into three classes: correctable, non-fatal, and fatal. Prior to discussing the PCI Express error reporting capabilities, including PCI-compatible mechanisms, a brief review of the PCI error handling is included as background information.

Two Methods of Interrupt Delivery

Interrupt delivery is conditionally optional for PCI Express devices. When a native PCI Express function does depend upon delivering interrupts to call its device driver, Message Signaled Interrupts (MSI) must be used. However, in the event that a device connecting to a PCI Express link cannot use MSIs (i.e., legacy devices), an alternate mechanism is defined. Both mechanisms are summarized below:

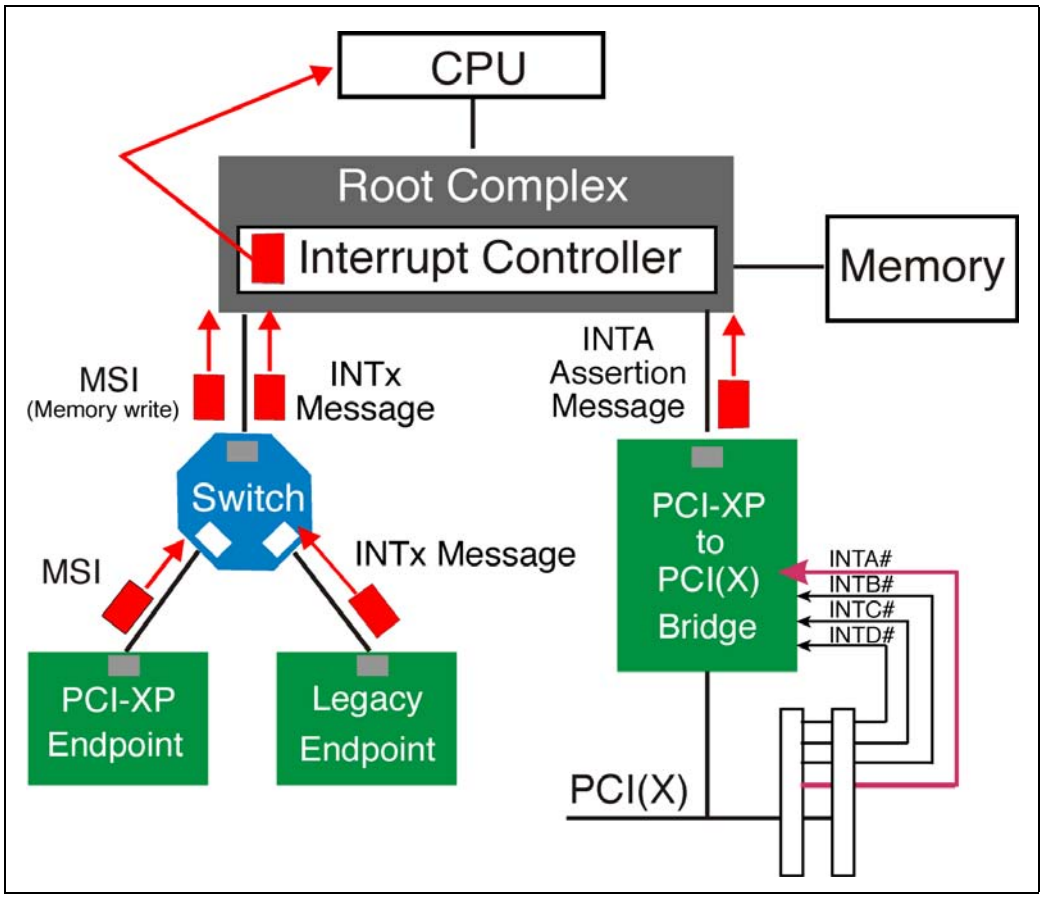
Native PCI Express Interrupt Delivery — PCI Express eliminates the need for sideband signals by using the Message Signaled Interrupt (MSI), first defined by the 2.2 version of the PCI Specification (as an optional mechanism) and later required by PCI-X devices. The term "Message Signaled Interrupt" can be misleading in the context of PCI Express because of possible confusion with PCI Express's "Message" transactions. A Message Signaled Interrupt is not a PCI Express Message, instead it is simply a Memory Write transaction. A memory write associated with an MSI can only be distinguished from other memory writes by the address locations they target, which are reserved by the system for Interrupt delivery.

Legacy PCI Interrupt Delivery — This mechanism supports devices that must use PCI-Compatible interrupt signaling (i.e., INTA#, INTB#, INTC#, and INTD#) defined for the PCI bus. Legacy functions use one of the interrupt lines to signal an interrupt. An INTx# signal is asserted to request interrupt service and deasserted when the interrupt service accesses a device-specific register, thereby indicating the interrupt is being serviced. PCI Express defines in-band messages that act as virtual INTx# wires, which target the interrupt controller located typically within the Root Complex.

Figure 9-1 illustrates the delivery of interrupts from three types of devices:

- Native PCI Express device — must use MSI delivery
- Legacy endpoint device — must support MSI and optionally support INTx messages. Such devices may be boot devices that must use legacy interrupts during boot, but once its driver loads MSIs are used.
- PCI Express-to-PCI (X) Bridge — must support INTx messages

Figure 9-1: Native PCI Express and Legacy PCI Interrupt Delivery



Message Signaled Interrupts

Message Signaled Interrupts (MSIs) are delivered to the Root Complex via memory write transactions. The MSI Capability register provides all the information that the device requires to signal MSIs. This register is set up by configuration software and includes the following information:

- Target memory address
- Data Value to be written to the specified address location
- The number of messages that can be encoded into the data

See "Description of 3DW And 4DW Memory Request Header Fields" on page 176 for a review of the Memory Write Transaction Header. Note that MSIs always have a data payload of 1DW.

The MSI Capability Register Set

A PCI Express function indicates its support for MSI via the MSI Capability registers. Each native PCI Express function must implement a single MSI register set within its own configuration space. Note that the PCI Express specification defines two register formats:

1. 64-bit memory addressing format (Figure 9-2 on page 332) — required by all native PCI Express devices and optionally implemented by Legacy endpoints.
2. 32-bit memory addressing format (Figure 9-3 on page 332) — optionally supported by Legacy endpoints.

Figure 9-2: 64-bit MSI Capability Register Format

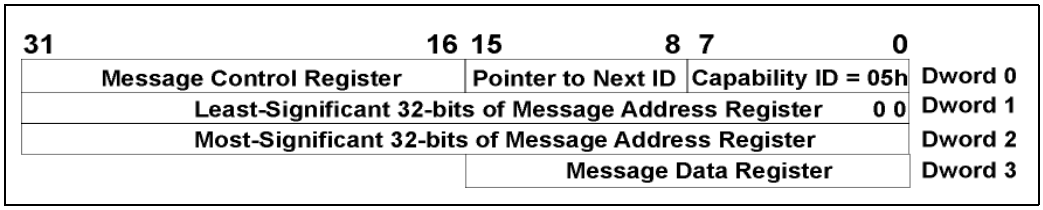
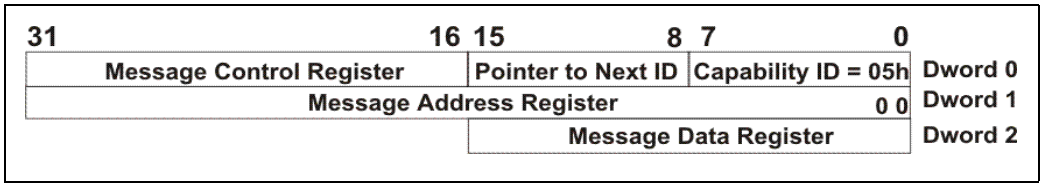


Figure 9-3: 32-bit MSI Capability Register Set Format



The following sections describe each field within the MSI registers.

Capability ID

The Capability ID that identifies the MSI register set is **05h**. This is a hardwired, read-only value.

Pointer To Next New Capability

The second byte of the register set either points to the next New Capability's register set or contains 00h if this is the end of the New Capabilities list. This is a hardwired, read-only value. If non-zero, it must be a dword-aligned value.

Message Control Register

Figure 9-4 on page 333 and Table 9-1 on page 333 illustrate the layout and usage of the Message Control register.

Figure 9-4: Message Control Register

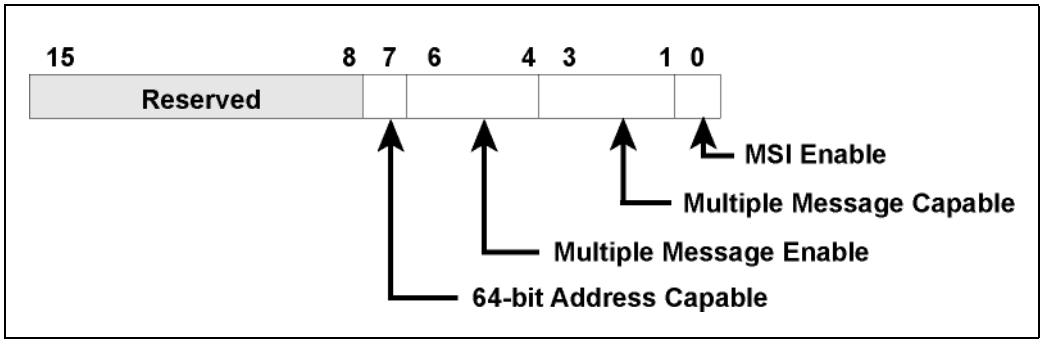


Table 9-1: Format and Usage of Message Control Register

Bit(s)	Field Name	Description
15:8	Reserved	Read-Only. Always zero.
7	64-bit Address Capable	Read-Only. • 0 = Function does not implement the upper 32-bits of the Message Address register and is incapable of generating a 64-bit memory address. • 1 = Function implements the upper 32-bits of the Message Address register and is capable of generating a 64-bit memory address.

Table 9-1: Format and Usage of Message Control Register (Continued)

Bit(s)	Field Name	Description
6:4	Multiple Message Enable	Read/Write. After system software reads the Multiple Message Capable field (see next row in this table) to determine how many messages are requested by the device, it programs a 3-bit value into this field indicating the actual number of messages allocated to the device. The number allocated can be equal to or less than the number actually requested. The state of this field after reset is 000b. The field is encoded as follows: **Value** — **Number of Messages Requested** 000b — 1 001b — 2 010b — 4 011b — 8 100b — 16 101b — 32 110b — Reserved 111b — Reserved
3:1	Multiple Message Capable	Read-Only. System software reads this field to determine how many messages the device would like allocated to it. The requested number of messages is a power of two, therefore a device that would like three messages must request that four messages be allocated to it. The field is encoded as follows: **Value** — **Number of Messages Requested** 000b — 1 001b — 2 010b — 4 011b — 8 100b — 16 101b — 32 110b — Reserved 111b — Reserved

10 Error Detection and Handling

The Previous Chapter

Native PCI Express devices that require interrupt support must use the Message Signaled Interrupt (MSI) mechanism defined originally in the PCI 2.2 version of the specification. The previous chapter detailed the MSI mechanism and also described the legacy support that permits virtualization of the PCI INTx signals required by devices such as PCI Express-to-PCI Bridges.

This Chapter

To this point it has been presumed that transactions traversing the fabric have not encountered any errors that cannot be corrected by hardware. This chapter discusses both correctable and non-correctable errors and discusses the mechanisms used to report them. The PCI Express architecture provides a rich set of error detection, reporting, and logging capabilities. PCI Express error reporting classifies errors into three classes: correctable, non-fatal, and fatal. PCI Express error reporting capabilities include PCI-compatible mechanisms, thus a brief review of the PCI error handling is included as background information.

The Next Chapter

The next chapter describes the Logical Physical Layer core logic. It describes how an outbound packet is processed before clocking the packet out differentially. The chapter also describes how an inbound packet arriving from the Link is processed and sent to the Data Link Layer. Sub-block functions of the Physical Layer such as Byte Striping and Un-Striping logic, Scrambler and De-Scrambler, 8b/10b Encoder and Decoder, Elastic Buffers are discussed, and more.

Background

The original PCI bus implementation provides for basic parity checks on each transaction as it passes between two devices residing on the same bus. When a transaction crosses a bridge, the bridge is involved in the parity checks at both the originating and destination busses. Any error detected is registered by the device that has detected the error and optionally reported. The PCI architecture provides a method for reporting the following types of errors:

- data parity errors — reported via the PERR# (Parity Error) signal
- data parity errors during multicast transactions (special cycles) — reported via the SERR# (System Error) signal
- address and command parity errors — reported via the SERR# signal
- other types of errors (e.g. device specific) — reported via SERR#

Errors reported via PERR# are considered potentially recoverable, whereas, errors reported via SERR# are considered unrecoverable. How the errors reported via PERR# are handled is left up to the implementer. Error handling may involve only hardware, device-specific software, or system software. Errors signaled via SERR# are reported to the system and handled by system software. (See MindShare's PCI System Architecture book for details.)

PCI-X uses the same error reporting signals as PCI, but defines specific error handling requirements depending on whether device-specific error handling software is present. If a device-specific error handler is not present, then all parity errors are reported via SERR#.

PCI-X 2.0 adds limited support for Error Correction Codes (ECC) designed to automatically detect and correct single-bit errors within the address or data. (See MindShare's PCI-X System Architecture book for details.)

Introduction to PCI Express Error Management

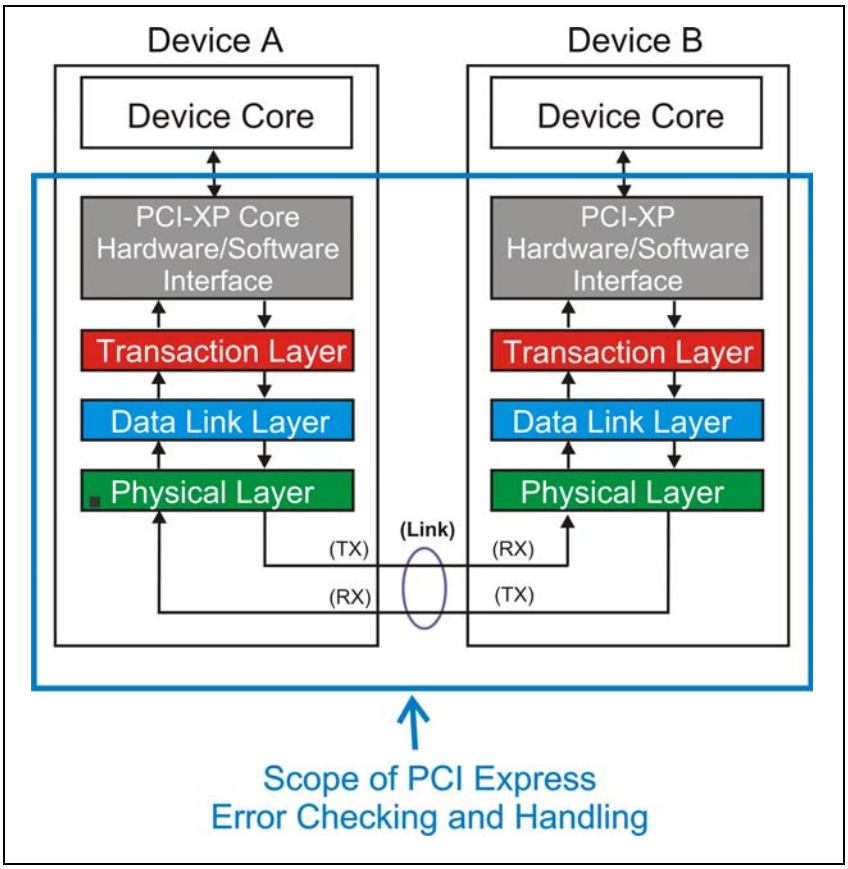
PCI Express defines a variety of mechanisms used for checking errors, reporting those errors and identifying the appropriate hardware and software elements for handling these errors.

PCI Express Error Checking Mechanisms

PCI Express error checking focuses on errors associated with the PCI Express interface and the delivery of transactions between the requester and completer functions. Figure 10-1 on page 357 illustrates the scope of the error checking that

is the focus of this chapter. Errors within a function that do not pertain to a given transaction are not reported through the error handling procedures defined by the PCI Express specification, and it is recommended that such errors be handled using proprietary methods that are reported via device-specific interrupts. Each layer of the PCI Express interface includes error checking capability as described in the following sections.

Figure 10-1: The Scope of PCI Express Error Checking and Reporting



Transaction Layer Errors

The transaction layer checks are performed only by the Requestor and Completer. Packets traversing switches do not perform any transaction layer checks. Checks performed at the transaction layer include:

- ECRC check failure (optional check based on end-to-end CRC)
- Malformed TLP (error in packet format)
- Completion Time-outs during split transactions
- Flow Control Protocol errors (optional)
- Unsupported Requests
- Data Corruption (reported as a poisoned packet)
- Completer Abort (optional)
- Unexpected Completion (completion does not match any Request pending completion)
- Receiver Overflow (optional check)

Data Link Layer Errors

Link layer error checks occur within a device involved in delivering the transaction between the requester and completer functions. This includes the requesting device, intermediate switches, and the completing device. Checks performed at the link layer include:

- LCRC check failure for TLPs
- Sequence Number check for TLP s
- LCRC check failure for DLLPs
- Replay Time-out
- Replay Number Rollover
- Data Link Layer Protocol errors

Physical Layer Errors

Physical layer error checks are also performed by all devices involved in delivering the transaction, including the requesting device, intermediate switches, and the completing device. Checks performed at the physical layer include:

- Receiver errors (optional)
- Training errors (optional)

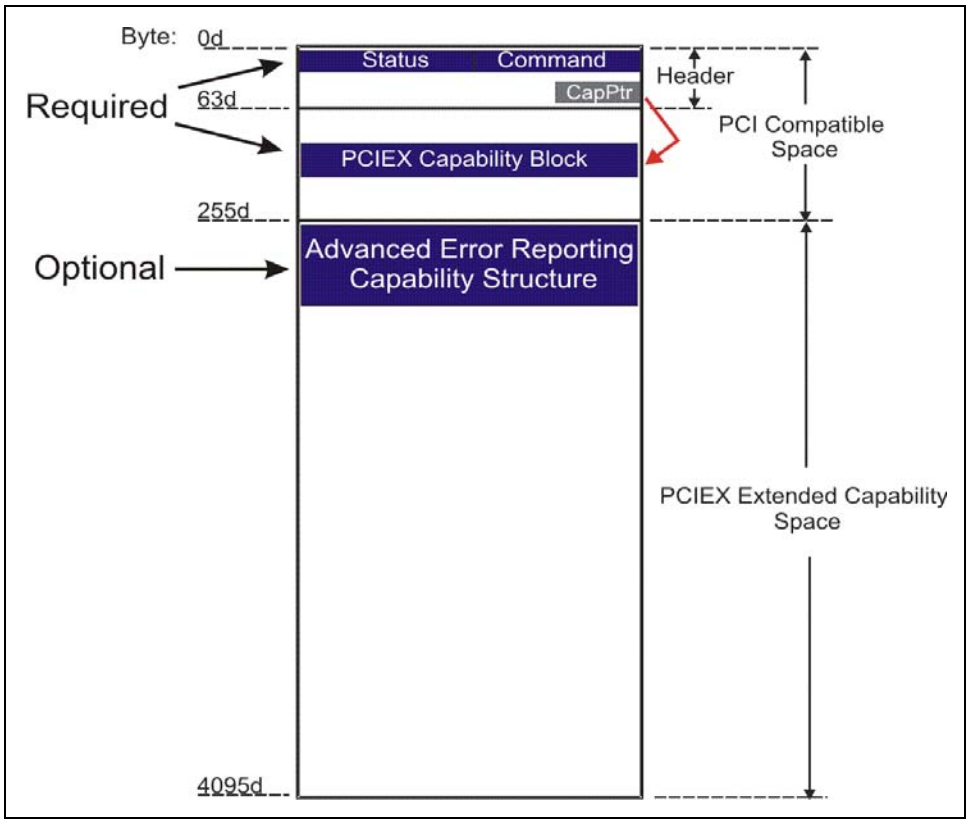
Error Reporting Mechanisms

PCI Express provides three mechanisms for establishing the error reporting policy. These mechanisms are controlled and reported through configuration registers mapped into three distinct regions of configuration space. (See Figure 10-2 on page 360.) The various error reporting features are enabled as follows:

- PCI-compatible Registers (required) — this error reporting mechanism provides backward compatibility with existing PCI compatible software and is enabled via the PCI configuration *Command Register*. This approach requires that PCI Express errors be mapped to PCI compatible error registers.
- PCI Express Capability Registers (required) — this mechanism is available only to software that has knowledge of PCI Express. This required error reporting is enabled via the PCI Express *Device Control Register* mapped within PCI-compatible configuration space.
- PCI Express Advanced Error Reporting Registers (optional) — this mechanism involves registers mapped into the extended configuration address space. PCI Express compatible software enables error reporting for individual errors via the *Error Mask Register.*

The specification refers to baseline (required) error reporting capabilities and advanced (optional) error reporting capabilities. The baseline error reporting mechanisms require access to the PCI-compatible registers and PCI Express Capability registers (bullets 1 and 2 above), while advanced error reporting (bullet 3) requires access to the Advanced Error Reporting registers that are mapped into extended configuration address space as illustrated in Figure 10-2. This chapter details all error reporting mechanisms.

Figure 10-2: Location of PCI Express Error-Related Configuration Registers



Error Handling Mechanisms

Errors are categorized into three classes that specify the severity of an error as listed below. Note also the specification defines the entity that should handle the error based on its severity:

- Correctable errors — handled by hardware
- Uncorrectable errors-nonfatal — handled by device-specific software
- Uncorrectable errors-fatal — handled by system software

11 Physical Layer Logic

The Previous Chapter

The previous chapter discussed both correctable and non-correctable errors and the mechanisms used to log and report them. Prior to discussing the PCI Express error reporting capabilities, a brief review of the PCI error handling was included as background information.

This Chapter

This chapter describes the Logical characteristics of the Physical Layer core logic. It describes how an outbound packet is processed before clocking the packet out differentially. The chapter also describes how an inbound packet arriving from the Link is processed and sent to the Data Link Layer. The chapter describes sub-block functions of the Physical Layer such as Byte Striping and Un-Striping logic, Scrambler and De-Scrambler, 8b/10b Encoder and Decoder, Elastic Buffers and more.

The Next Chapter

The next chapter describes the electrical characteristics of the Physical Layer. It describes the analog characteristics of the differential drivers and receivers that connect a PCI Express device to the Link.

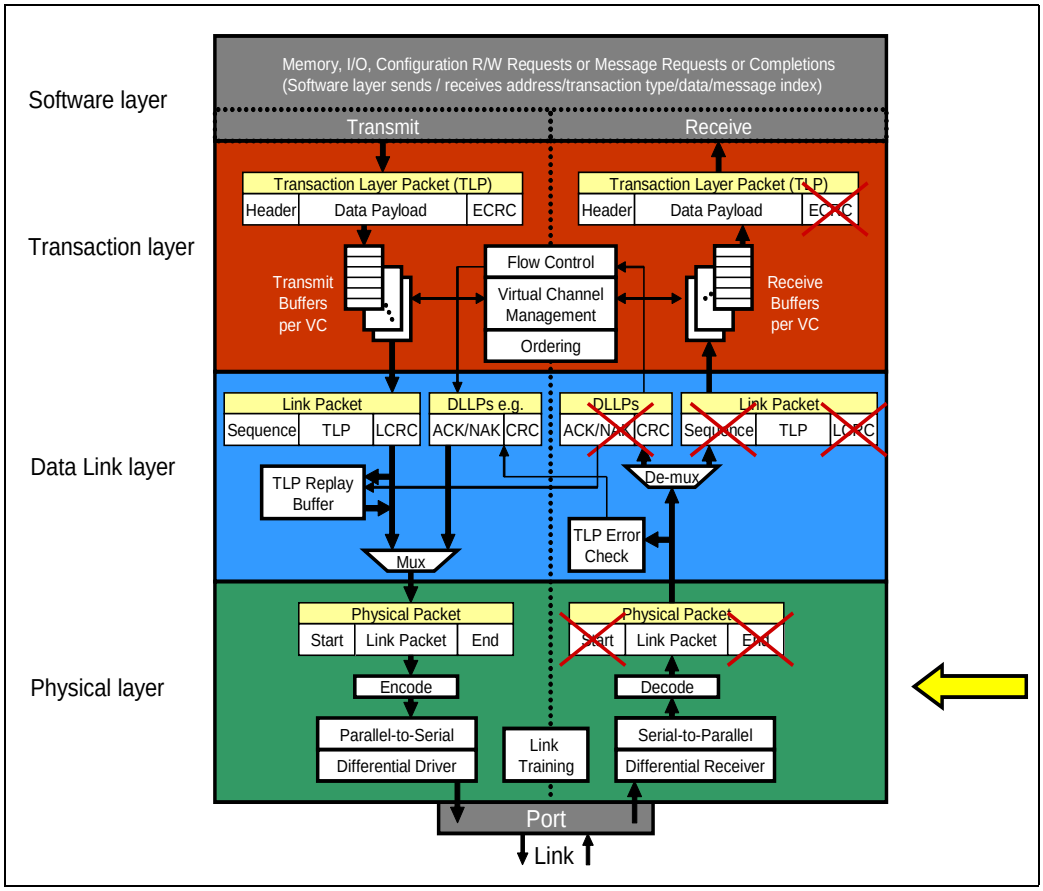
Physical Layer Overview

The Physical Layer shown in Figure 11-1 on page 398 connects to the Link on one side and interfaces to the Data Link Layer on the other side. The Physical Layer processes outbound packets before transmission to the Link and processes inbound packets received from the Link. The two sections of the Physical Layer associated with transmission and reception of packets are referred to as the transmit logic and the receive logic throughout this chapter.

The transmit logic of the Physical Layer essentially processes packets arriving from the Data Link Layer, then converts them into a serial bit stream. The bit stream is clocked out at 2.5 Gbits/s/Lane onto the Link.

The receive logic clocks in a serial bit stream arriving on the Lanes of the Link with a clock that is recovered from the incoming bit stream. The receive logic converts the serial bit steam into a parallel symbol stream, processes the incoming symbols, assembles packets and sends them to the Data Link Layer.

Figure 11-1: Physical Layer

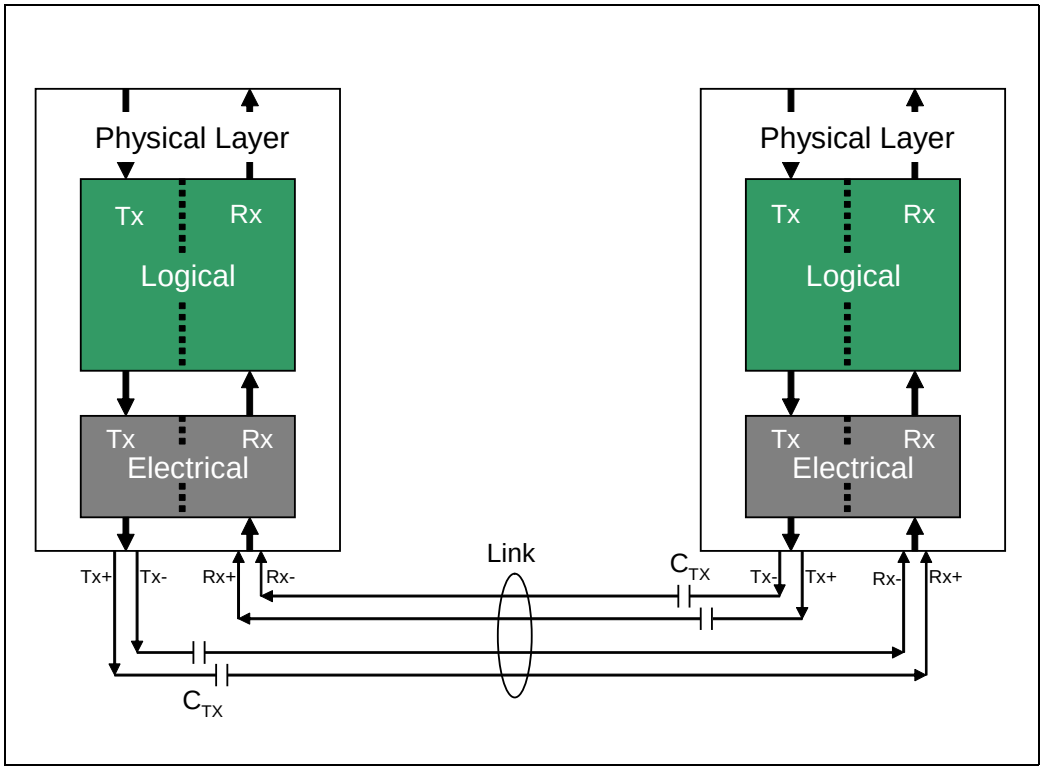


In the future, data rates per Lane are expected to go to 5 Gbits/s, 10 Gbits/s and beyond. When this happens, an existing design can be adapted to the higher data rates by redesigning the Physical Layer while maximizing reuse of the Data Link Layer, Transaction Layer and Device Core/Software Layer. The Phys-

ical Layer may be designed as a standalone entity separate from the Data Link Layer and Transaction Layer. This allows a design to be migrated to higher data rates or even to an optical implementation if such a Physical Layer is supported in the future.

Two sub-blocks make up the Physical Layer. These are the logical Physical Layer and the electrical Physical Layer as shown in Figure 11-2. This chapter describes the logical sub-block, and the next chapter describes the electrical sub-block. Both sub-blocks are split into transmit logic and receive logic (independent of each other) which allow dual simplex communication.

Figure 11-2: Logical and Electrical Sub-Blocks of the Physical Layer



Disclaimer

To facilitate description of the Physical Layer functionality, an example implementation is described that is not necessarily the implementation assumed by the specification nor is a designer compelled to implement a Physical Layer in such a manner. A designer may implement the Physical Layer in any manner that is compliant with the functionality expected by the PCI Express specification.

Transmit Logic Overview

Figure 11-3 on page 401 shows the elements that make up the transmit logic:

- a multiplexer (mux),
- byte striping logic (only necessary if the link implements more than one data lane),
- scramblers,
- 8b/10b encoders,
- and parallel-to-serial converters.

TLPs and DLLPs from the Data Link layer are clocked into a Tx (transmit) Buffer. With the aid of a multiplexer, the Physical Layer frames the TLPs or DLLPs with Start and End characters. These characters are framing symbols which the receiver device uses to detect start and end of packet.

The framed packet is sent to the **Byte Striping logic** which multiplexes the bytes of the packet onto the Lanes. One byte of the packet is transferred on one Lane, the next byte on the next Lane and so on for the available Lanes.

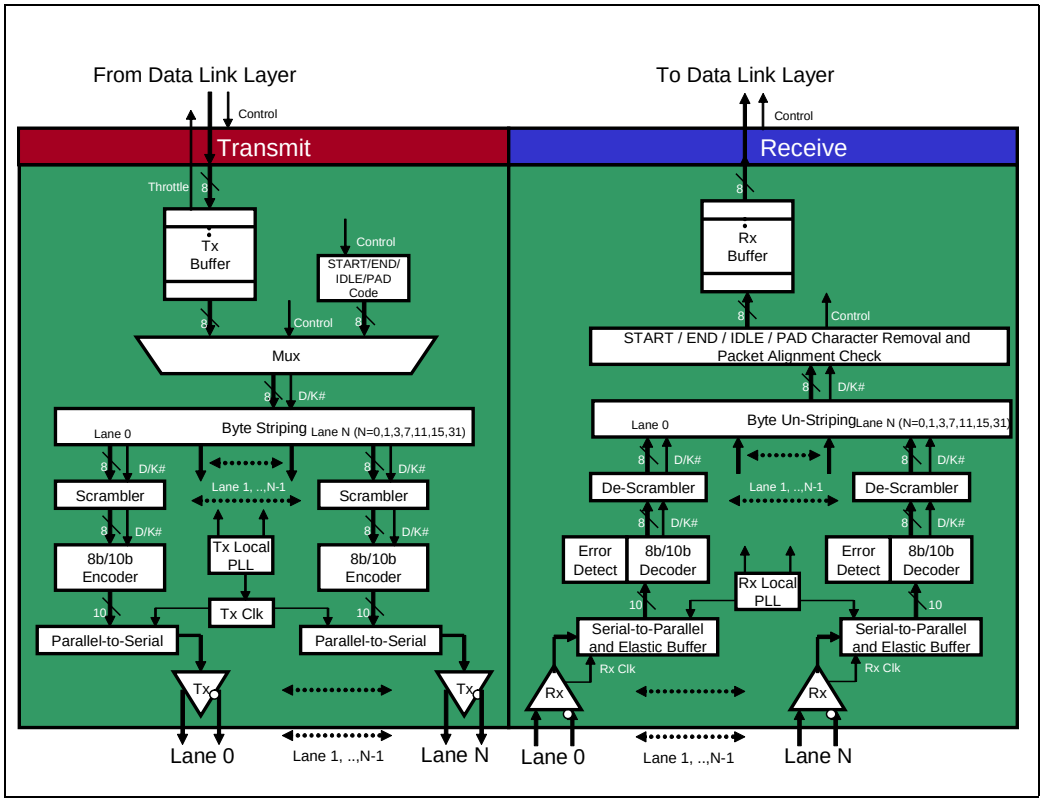
The **Scrambler** uses an algorithm to pseudo-randomly scramble each byte of the packet. The Start and End framing bytes are not scrambled. Scrambling eliminates repetitive patterns in the bit stream. Repetitive patterns result in large amounts of energy concentrated in discrete frequencies which leads to significant EMI noise generation. Scrambling spreads energy over a frequency range, hence minimizing average EMI noise generated.

The scrambled 8-bit characters (8b characters) are encoded into 10-bit symbols (10b symbols) by the **8b/10b Encoder** logic. And yes, there is a 25% loss in transmission performance due to the expansion of each byte into a 10-bit character. A **Character** is defined as the 8-bit un-encoded byte of a packet. A **Symbol** is defined as the 10-bit encoded equivalent of the 8-bit character. The purpose of

8b/10b Encoding the packet characters is primarily to create sufficient 1-to-0 and 0-to-1 transition density in the bit stream so that the receiver can re-create a receive clock with the aid of a **receiver Phase Lock Loop (PLL)**. Note that the clock used to clock the serial data bit stream out of the transmitter is not itself transmitted onto the wire. Rather, the receive clock is used to clock in an inbound packet.

The 10b symbols are converted to a serial bit stream by the **Parallel-to-Serial converter**. This logic uses a 2.5 GHz clock to serially clock the packets out on each Lane. The serial bit stream is sent to the electrical sub-block which **differentially transmits** the packet onto each Lane of the Link.

Figure 11-3: Physical Layer Details



Receive Logic Overview

Figure 11-3 shows the elements that make up the receiver logic:

- receive PLL,
- serial-to-parallel converter,
- elastic buffer,
- 8b/10b decoder,
- de-scrambler,
- byte un-striping logic (only necessary if the link implements more than one data lane),
- control character removal circuit,
- and a packet receive buffer.

As the data bit stream is received, the **receiver PLL** is synchronized to the clock frequency with which the packet was clocked out of the remote transmitter device. The transitions in the incoming serial bit stream are used to re-synchronize the PPL circuitry and maintain bit and symbol lock while generating a clock recovered from the data bit stream. The serial-to-parallel converter is clocked by the recovered clock and outputs 10b symbols.

The 10b symbols are clocked into the **Elastic Buffer** using the recovered clock associated with the receiver PLL. The Elastic Buffer is used for clock tolerance compensation; i.e. the Elastic Buffer is used to adjust for minor clock frequency variation between the recovered clock used to clock the incoming bit stream into the Elastic Buffer and the locally-generated clock associated that is used to clock data out of the Elastic Buffer.

The 10b symbols are converted back to 8b characters by the **8b/10b Decoder**. The Start and End characters that frame a packet are eliminated. The 8b/10b Decoder also looks for errors in the incoming 10b symbols. For example, error detection logic can check for invalid 10b symbols or detect a missing Start or End character.

The **De-Scrambler** reproduces the de-scrambled packet stream from the incoming scrambled packet stream. The De-Scrambler implements the inverse of the algorithm implemented in the transmitter Scrambler.

The bytes from each Lane are **un-striped** to form a serial byte stream that is loaded into the **receive buffer** to feed to the Data Link layer.

12 Electrical Physical Layer

The Previous Chapter

The previous chapter described:

- The logical Physical Layer core logic and how an outbound packet is processed before clocking the packet out differentially.
- How an inbound packet arriving from the Link is processed and sent to the Data Link Layer.
- Sub-block functions of the Physical Layer such as Byte Striping and unstriping logic, Scrambler and De-Scrambler, 8b/10b Encoder and decoder, Elastic Buffers and more.

This Chapter

This chapter describes the Physical Layer's electrical interface to the link. It describes the analog characteristics of the differential drivers and receivers that connect a PCI Express device to the Link. Timing and driver/receiver parameters are documented here.

The Next Chapter

The next chapter describes the three types of reset, namely: cold reset, warm reset and hot reset. It also describes the usage of a side-band reset signal called PERST#. The effect of reset on devices and the system is described.

Electrical Physical Layer Overview

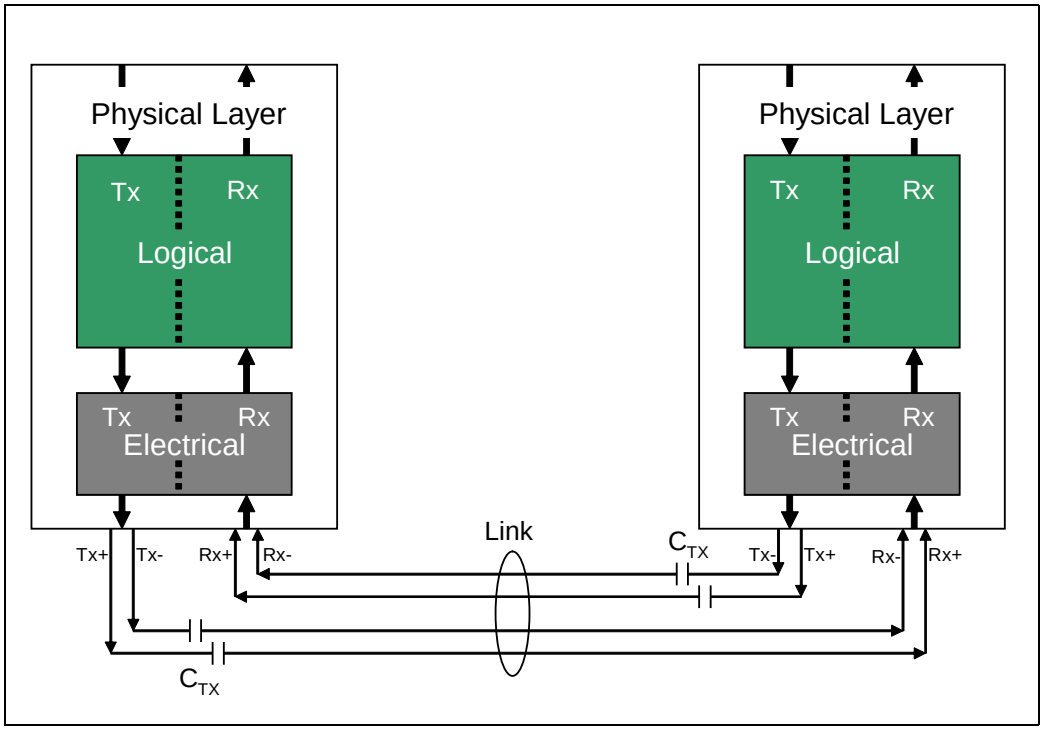
The electrical sub-block associated with each lane (see Figure 12-1 on page 454) provides the physical interface to the Link. This sub-block contains differential drivers (transmitters) and differential receivers (receivers). The transmitter serializes outbound symbols on each Lane and converts the bit stream to electrical

signals that have an embedded clock. The receiver detects electrical signaling on each Lane and generates a serial bit stream that it de-serializes into symbols, and supplies the symbol stream to the logical Physical Layer along with the clock recovered from the inbound serial bit stream.

In the future, this sub-block could be redesigned to support a cable interface or an optical (i.e., fiber) interface.

In addition, the electrical Physical Layer contains a Phase Lock Loop (PLL) that drives the Serializer in the transmitter and a receiver PLL that is sync'd to the transitions in the incoming serial symbol stream.

Figure 12-1: Electrical Sub-Block of the Physical Layer



When the Link is in the L0 full-on state, the differential drivers drive the differential voltage associated with a logical 1 and logical 0 while driving the correct DC common mode voltage. The receivers sense differential voltages that indicate a logical 1 or 0 and, inaddition, can sense the electrical idle state of the Link. An eye diagram clearly illustrates the electrical characteristics of a driver and receiver and addresses signaling voltage levels, skew and jitter issues.

The electrical Physical Layer is responsible for placing the differential drivers, differential receivers, and the Link in the correct state when the Link is placed in a low power state such as L0s, L1, or L2. While in the L2 low power state, a device can signal a wake-up event upstream via a Beacon signaling mechanism.

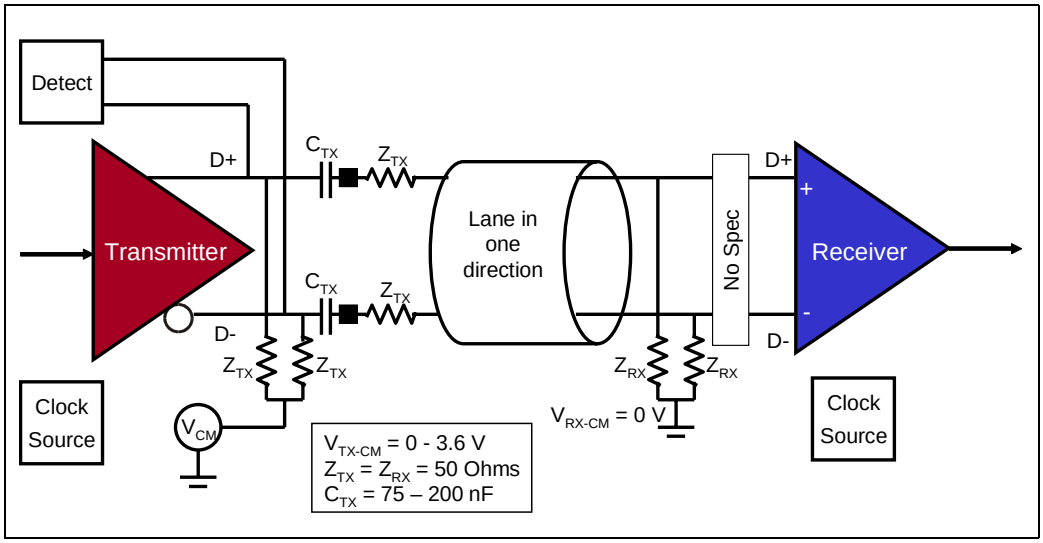
The differential drivers support signal de-emphasis (or pre-emphasis; see "De-Emphasis (or Pre-Emphasis)" on page 466) to help reduce the bit error rate (BER)—especially on a lossy Link.

The drivers and receivers are short-circuit tolerant, making them ideally suited for hot insertion and removal events. The Link connecting two devices is AC coupled. A capacitor at the transmitter side of the Link DC de-couples it from the receiver. As a result, two devices at opposite ends of a Link can have their own ground and power planes. See Figure 12-1 on page 454 for the capacitor (C_{TX}) placement on the Link.

High Speed Electrical Signaling

Refer to Figure 12-2. High-speed LVDS (Low-Voltage Differential Signaling) electrical signaling is used in driver and receiver implementations. Drivers and receivers from different manufacturers must be inter-operable and may be designed to be hot-pluggable. A standard FR4 board can be used to route the Link wires. The following sections describe the electrical characteristics of the driver, receiver, and the Link represented in the Figure.

Figure 12-2: Differential Transmitter/Receiver



Clock Requirements

General

The transmitter clocks data out at 2.5Gbits/s. The clock used to do so must be accurate to +/- 300 ppm of the center frequency. It is allowed to skew a maximum of 1 clock every 1666 clocks. The two devices at the opposite ends of a Link could have their transmit clocks out of phase by as much as 600 ppm.

A device may derive its clock from an external clock source. The system board supplies a 100 MHz clock that is made available to devices on the system board as well as to add-in cards via the connector. With the aid of PLLs, a device may generate its required clocks from this 100 MHz clock.

Spread Spectrum Clocking (SSC)

Spread spectrum clocking is a technique used to modulate the clock frequency slowly so as to reduce EMI radiated noise at the center frequency of the clock. With SSC, the radiated energy does not produce a noise spike at 2.5GHz because the radiated energy is spread over a small frequency range around 2.5GHz.

SCC is not required by the specification. However, if supported, the following rules apply:

- The clock can be modulated by +0% to -0.5% from nominal a frequency of 2.5GHz.
- The modulation rate must be between 30KHz and 33KHz.
- The +/- 300 ppm requirement for clock frequency accuracy still holds. Further, the maximum of 600 ppm frequency variation between the two devices at opposite ends of a Link also remains true. This almost certainly imposes a requirement that the two devices at opposite ends of the Link be driven from the same clock source when the clock is modulated with SSC.

Impedance and Termination

The characteristic impedance of the Link is 100 Ohms differential (nominal), while single-ended DC common mode impedance is 50 Ohms. This impedance is matched to the transmitter and receiver impedances.

Transmitter Impedance Requirements

Transmitters must meet the $Z_{TX\text{-}DIFF\text{-}DC}$ (see Table 12-1 on page 477) parameters anytime differential signals are transmitted during the full-on L0 power state.

When a differential signal is not driven (e.g., in the lower power states), the transmitter may keep its output impedance at a minimum $Z_{TX\text{-}DC}$ (see Table 12-1 on page 477) of 40 Ohms, but may also place the driver in a high impedance state. Placing a driver in the high impedance state may be helpful while in L0s or L1 low power states to help reduce power drain in these states.

Receiver Impedance Requirements

The receiver is required to meet the $Z_{RX\text{-}DIFF\text{-}DC}$ (see Table 12-2 on page 480) parameter of 100 Ohms anytime differential signals are transmitted during the full-on L0 power state, as well as in all other lower power states wherein adequate power is provided to the device. A receiver is excluded from this impedance requirement when the device is powered down (e.g., in the L2 and L3 power states and during Fundamental Reset).

When a receiver is powered down to the L2 or L3 state, or during Fundamental Reset, its receiver goes to the high impedance state and must meet the $Z_{RX\text{-}HIGH\text{-}IMP\text{-}DC}$ parameter of 200 KOhms minimum (see Table 12-2 on page 480).

DC Common Mode Voltages

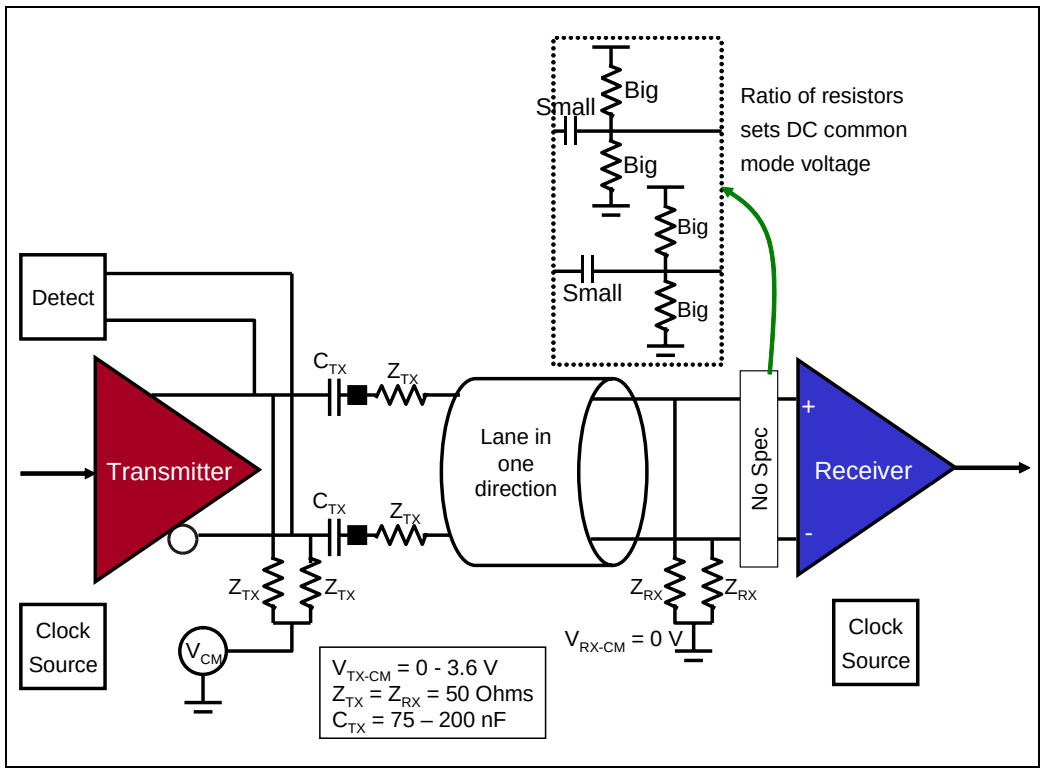
Transmitter DC Common Mode Voltage

Once driven after power-on and during the Detect state of Link training, the transmitter DC common mode voltage $V_{TX\text{-}DC\text{-}CM}$ (see Table 12-1 on page 477) must remain at the same voltage. The common mode voltage is turned off only when the transmitter is placed in the L2 or L3 low power state, during which main power to the device is removed. A designer can choose any common mode voltage in the range of 0V to 3.6V.

Receiver DC Common Mode Voltage

The receiver is DC de-coupled from the transmitter by a capacitor. This allows the receiver to have its own DC common mode voltage. This voltage is specified at 0V. The specification is unclear about the meaning of this 0V receiver DC common mode voltage requirement and does not require the common mode voltage to be 0V at the input to the receiver differential amplifier. Rather, a simple bias voltage network allows the receiver to operate at optimal common mode. See Figure 12-3 on page 458.

Figure 12-3: Receiver DC Common Mode Voltage Requirement



ESD and Short Circuit Requirements

All signals and power pins must withstand (without damage) a 2000V Electro-Static Discharge (ESD) using the human body model and 500V using the charged device model. For more details on this topic, see the JEDEC JESE22-A114-A specification.

The ESD requirement not only protects against electro-static damage, but facilitates support of surprise hot insertion and removal events. Transmitters and receivers are also required to be short-circuit tolerant. They must be able to withstand sustained short-circuit currents (on D+ or D- to ground) of $I_{TX-SHORT}$ (see Table 12-2 on page 480) in the order of 90mA (the maximum current a transmitter is required to provide).

13 System Reset

The Previous Chapter

The previous chapter describes the Electrical Physical Layer. It describes the analog characteristics of the differential drivers and receivers that connect a PCI Express device to the Link. Timing and driver/receiver parameters are documented in that chapter.

This Chapter

This chapter describes 3 types of system reset generation capabilities: cold reset, warm reset and hot reset. The chapter also describes the usage of a side-band reset signal called PERST#. It describes the usage of the TS1 Ordered-Set to generate an in-band Hot Reset. The effect of reset on a device and system is described.

The Next Chapter

The next chapter describes the function of the Link Training and Status State Machine (LTSSM) of the Physical Layer. The chapter describes the initialization process of the Link from Power-On or Reset, until the full-on L0 state, where traffic on the Link can begin. In addition, the chapter describes the lower power management states L0s, L1, L2, L3 and briefly describes entry and exit procedure to/from these states.

Two Categories of System Reset

The PCI Express specification describes two reset generation mechanisms. The first mechanism is a system generated reset referred to as **Fundamental Reset**. The second mechanism is an In-band Reset (communicated downstream via the Link from one device to another) referred to as the **Hot Reset**.

Fundamental Reset

Fundamental Reset causes a device's state machines, hardware logic, port states and configuration registers (except sticky registers of a device that can draw valid V_{aux}) to initialize to their default conditions.

There are two types of Fundamental Reset:

- **Cold Reset**. This is a reset generated as a result of application of main power to the system.
- **Warm Reset.** Triggered by hardware without the removal and re-application of main power. A Warm Reset could be triggered due to toggling of the system 'POWERGOOD' signal with the system power stable. The mechanism for generating a Warm Reset is not defined by specification. It is up to the system designer to optionally provide a mechanism to generate a Warm Reset.

When Fundamental Reset is asserted:

- The receiver terminations are required to meet the $Z_{RX\text{-}HIGH\text{-}IMP\text{-}DC}$ parameter of 200 kOhms minimum (see Table 12-2 on page 480).
- The transmitter terminations are required to meet the output impedance at minimum $Z_{TX\text{-}DC}$ (see Table 12-1 on page 477) of 40 Ohms, but may place the driver in a high impedance state.
- The transmitter holds a constant DC common mode voltage between 0 V and 3.6 V.

After Fundamental Reset Exit:

- The receiver must re-enable its receiver terminations $Z_{RX\text{-}DIFF\text{-}DC}$ (see Table 12-2 on page 480) of 100 Ohms within 5 ms of Fundamental Reset exit. The receiver is now ready to detect electrical signaling on the Link.
- After Fundamental Reset exit, the Link Training state machine enters the 'Detect' state and the transmitter is ready to detect the presence of a receiver at the other end of the Link.
- The transmitter holds a constant DC common mode voltage between 0 V and 3.6 V.

Methods of Signaling Fundamental Reset

Fundamental Reset may be signaled via an auxiliary side-band signal called PERST# (PCI Express Reset, asserted low). When PERST# is not provided to an add-in card or component, Fundamental Reset is generated autonomously by the component or add-in card.

Below is a description of the two mechanisms of Fundamental Reset generation.

PERST# Type Fundamental Reset Generation. A central resource device, e.g. a chipset, in the PCI Express system provides this source of reset. For example, the IO Controller Hub (ICH) chip in Figure 13-1 on page 490 may generate PERST#. The system power supply (not shown in figure) generates a 'POWERGOOD' signal once main power is turned on and stable. The ICH Reset logic in-turn uses this signal to assert PERST# when POWERGOOD (asserted High) is deasserted. If power is cycled, POWERGOOD toggles and causes PERST# to assert and deassert. This is the Cold Reset. If the system provides a method of toggling POWERGOOD without cycling through power (as via a button on the chassis) then also PERST# asserts and deasserts. This is the Warm Reset.

The PERST# signal feeds all PCI Express devices on the motherboard including the connectors and graphics controller. Devices may choose to use PERST# but are not required to use it as the source of reset.

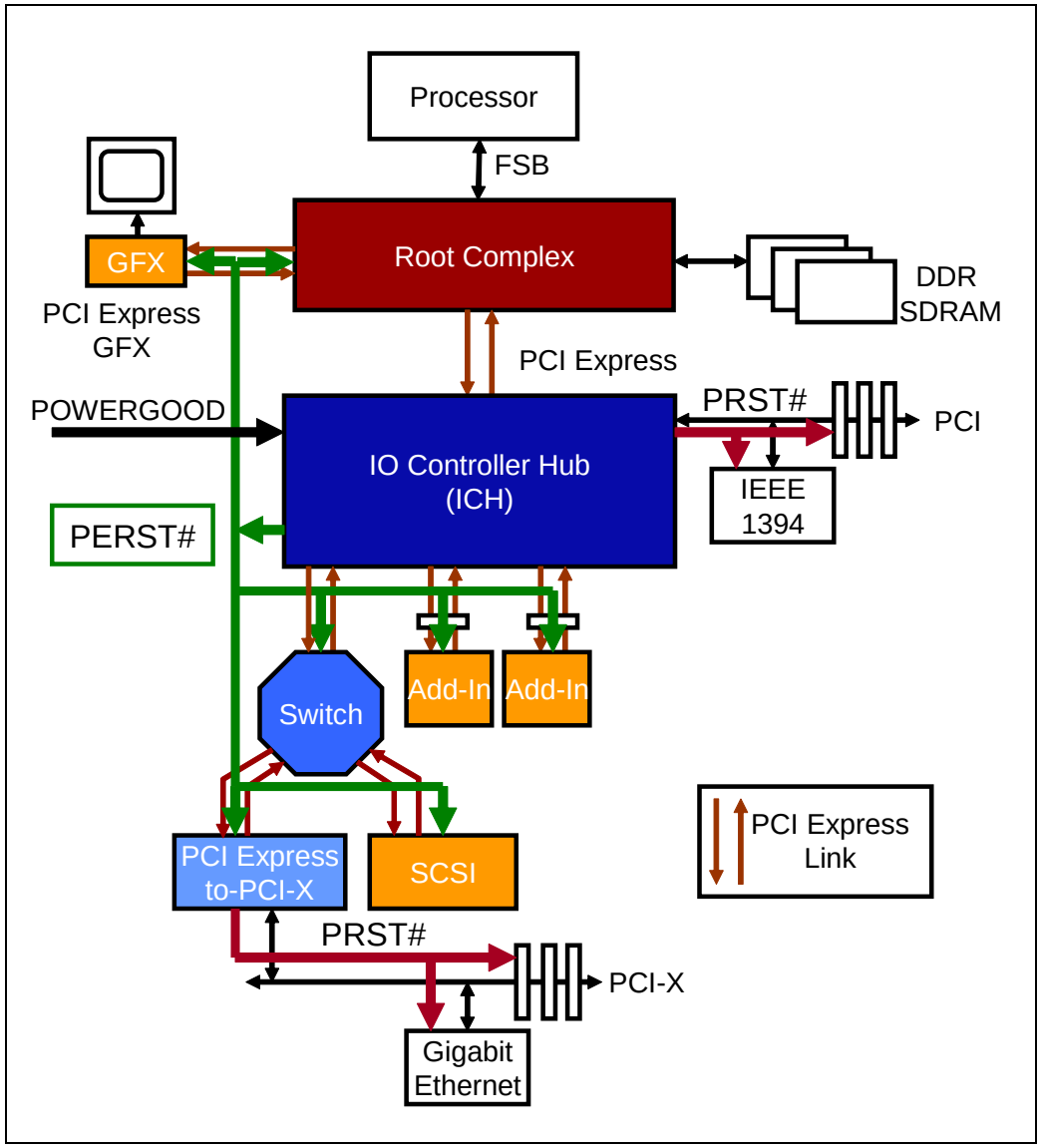
The PERST# signal also feeds the PCI Express-to-PCI-X bridge shown in the figure. The bridge forwards this reset to the PCI-X bus as PCI-X bus RST#. ICH also generates PRST# for the PCI bus.

Autonomous Method of Fundamental Reset Generation. A device can be designed to generate its own Fundamental Reset upon detection of application (or re-application) of main power. The specification does not describe the mechanism for doing so. The self reset generation mechanism can be built into the device or may be designed as external logic, for example, on a add-in card that detects Power-On and generates a local reset to the device.

The device must also generate an autonomous Fundamental Reset if it detects its power go outside of the limits specified.

A device should support the autonomous method of triggering a Fundamental Reset given that the specification is not clear about requirement of system PERST# support.

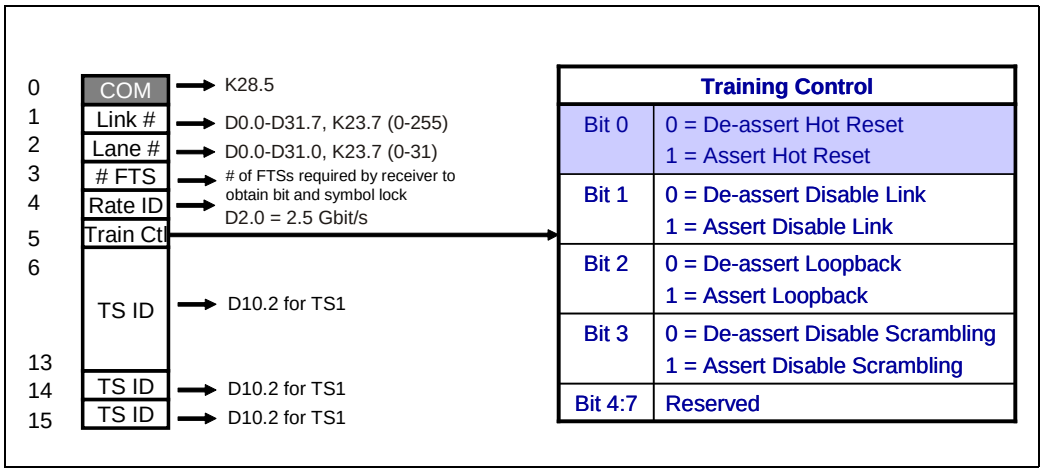
Figure 13-1: PERST# Generation



In-Band Reset or Hot Reset

Hot Reset is propagated in-band via the transmission of TS1 Ordered-Sets (shown in Figure 13-2) with bit 0 of symbol 5 in the TS1 Ordered-Set asserted. The TS1 Ordered-Set is transmitted on all Lanes with the correct Link # and Lane# symbols. These TS1 Ordered-Sets are continuously transmitted for 2 ms. Both transmitter and receiver of Hot Reset end up in the detect state (see "Hot Reset State" on page 544). Hot Reset, in general, is a software generated reset.

Figure 13-2: TS1 Ordered-Set Showing the Hot Reset Bit



Hot Reset is propagated downstream. Hot Reset is not propagated upstream. This means that only the Root Complex and Switches are able to generate Hot Reset. Endpoints do not generate Hot Reset. A switch that receives a Hot Reset TS1 Ordered-Set on its upstream port must pass it to all its downstream ports. In addition, the switch resets itself. All devices downstream of a switch that receive the Hot Reset TS1 Ordered-Set will reset themselves.

Response to Receiving a Hot Reset Command

When a device receives a Hot Reset command:

- It goes to the 'Detect' Link State (via the Recovery and Hot Reset state) of the Link Training state machine and starts the Link training process, followed by initialization of VC0.
- Its state machines, hardware logic, port states and configuration registers (except sticky registers) initialize to their default conditions.

Switches Generate Hot Reset on Their Downstream Ports

The following are a list of bullets that indicate when a switch generates a Hot Reset on ALL its downstream ports:

- Switch receives a Hot Reset on its upstream port
- The Data Link Layer of the switch upstream port reports a DL_Down state. This state occurs when the upstream port has been disconnected or when the upstream port has lost connection with an upstream device due to an error that is not recoverable by the Physical Layer and Data Link Layer.
- Software sets the 'Secondary Bus Reset' bit of the Bridge Control configuration register associated with the upstream port.

Bridges Forward Hot Reset to the Secondary Bus

If a bridge such as a PCI Express-to-PCI(-X) bridge detects a Hot Reset on its upstream port, it must assert the PRST# signal on its secondary PCI(-X) bus.

How Does Software Tell a Device (e.g. Switch or Root Complex) to Generate Hot Reset?

Software tells a root complex or switch to generate a Hot Reset on a specific port by writing a 1 followed by 0 to the 'Secondary Bus Reset' bit in the Bridge Control register of that associated port's configuration header. See Figure 13-3 on page 493 for the location of this bit. Consider the example shown in Figure 13-4 on page 494. Software writes a 1 to the 'Secondary Bus Reset' register of Switch A's downstream left side port. Switch A generates a Hot Reset on that port by forwarding TS1 Ordered-Sets with the Hot Reset bit set. Switch A does not generate a Hot Reset on its right side port. Switch B receives this Hot Reset on its upstream port and forwards it on all downstream ports to the two endpoints.

If software writes to the 'Secondary Bus Reset' bit of the switch's upstream port, then the switch generates a Hot Reset on ALL its downstream ports. Consider the example shown in Figure 13-5 on page 495. Software writes a 1 to the 'Secondary Bus Reset' register of Switch C's upstream port. Switch C generates a Hot Reset on ALL downstream ports by forwarding TS1 Ordered-Sets with the Hot Reset bit set on both ports. The PCI Express-to-PCI bridge receives this Hot Reset and forwards it on to the PCI bus by asserting PRST#.

A device is in the L0 state when the 'Secondary Bus Reset' bit is set. The device (upstream device) then goes through the Recovery state of the LTSSM (see "Recovery State" on page 532) before it generates the TS1 Ordered-Sets with the Hot Reset bit set and then enters the Hot Reset state (see "Hot Reset State" on

14 Link Initialization & Training

The Previous Chapter

The previous chapter described three types of system reset generation capabilities: cold reset, warm reset and hot reset. The chapter also described the usage of the side-band reset signal PERST#. The effect of reset on a device and system was described.

This Chapter

This chapter describes the function of the Link Training and Status State Machine (LTSSM) of the Physical Layer. The chapter describes the initialization process of the Link from Power-On or Reset, until the full-on L0 state, where traffic on the Link can begin. In addition, the chapter describes the lower power management states L0s, L1, L2, L3 and briefly describes entry and exit procedure to/from these states.

The Next Chapter

The next chapter describes the mechanical form factor for the PCI Express connector and add-in card. Different slot form factors are defined to support x1, x4, x8 and x16 Lane widths. In addition, the next chapter describes the Mini PCI Express form factor which targets the mobile market, Server IO Module (SIOM) form factor which targets the workstation and server market, and the NEWCARD form factor which targets both mobile and desktop markets.

Link Initialization and Training Overview

General

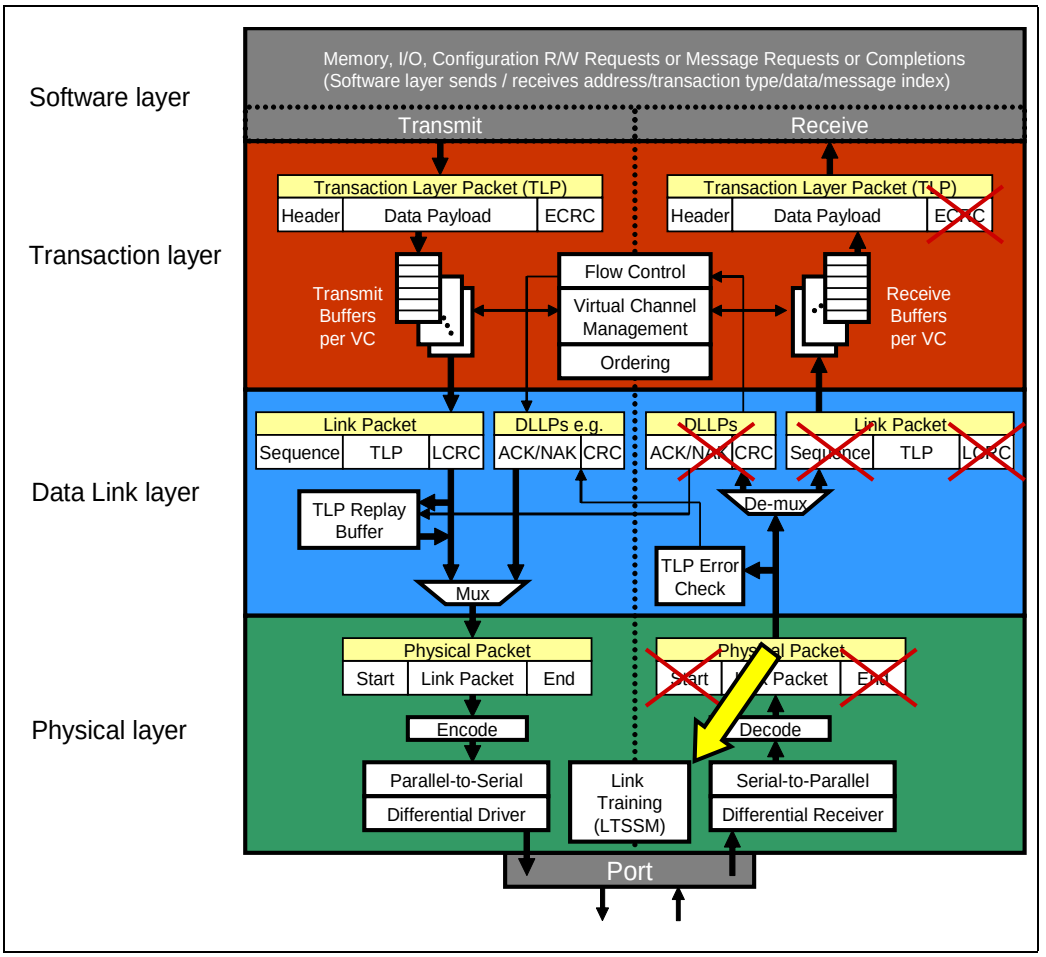
Link initialization and training is a Physical Layer control process that configures and initializes a device's Physical Layer, port, and associated Link so that normal packet traffic can proceed on the Link. This process is automatically initiated after reset without any software involvement. A sub-set of the Link training and initialization process, referred to as Link re-training, is initiated automatically as a result of a wakeup event from a low power mode, or due to an error condition that renders the Link inoperable. The Link Training and Status State Machine (LTSSM) is the Physical Layer sub-block responsible for the Link training and initialization process (see Figure 14-1).X

A receiver may optionally check for violations of the Link training and initialization protocol. If such an error occurs, it may be reported as a 'Link Training Error' to the error reporting logic (see "Link Errors" on page 379).

The following are configured during the Link training and initialization process:

- **Link Width** is established and set. Two devices with a different number of port Lanes may be connected. For example, one device with a x2 port may be connected to a device with a x4 port. During Link training and initialization, the Physical Layer of both devices determines and sets the Link width to the minimum Lane width of the two (i.e., x2). Other Link negotiated behaviors include Lane reversal, splitting of ports into multiple Links, and the configuration of a cross-Link.
- **Lane Reversal** on a multi-Lane device's port (if reversal is required). The Lanes on a device's port are numbered by design. When wiring up a Link to connect two devices, a board designer should match up the lane numbers of each device's port so that Lane 0 of one device's port connects to Lane 0 of the remote device's port, Lane *n* to Lane *n* of the remote device's port, and so on.

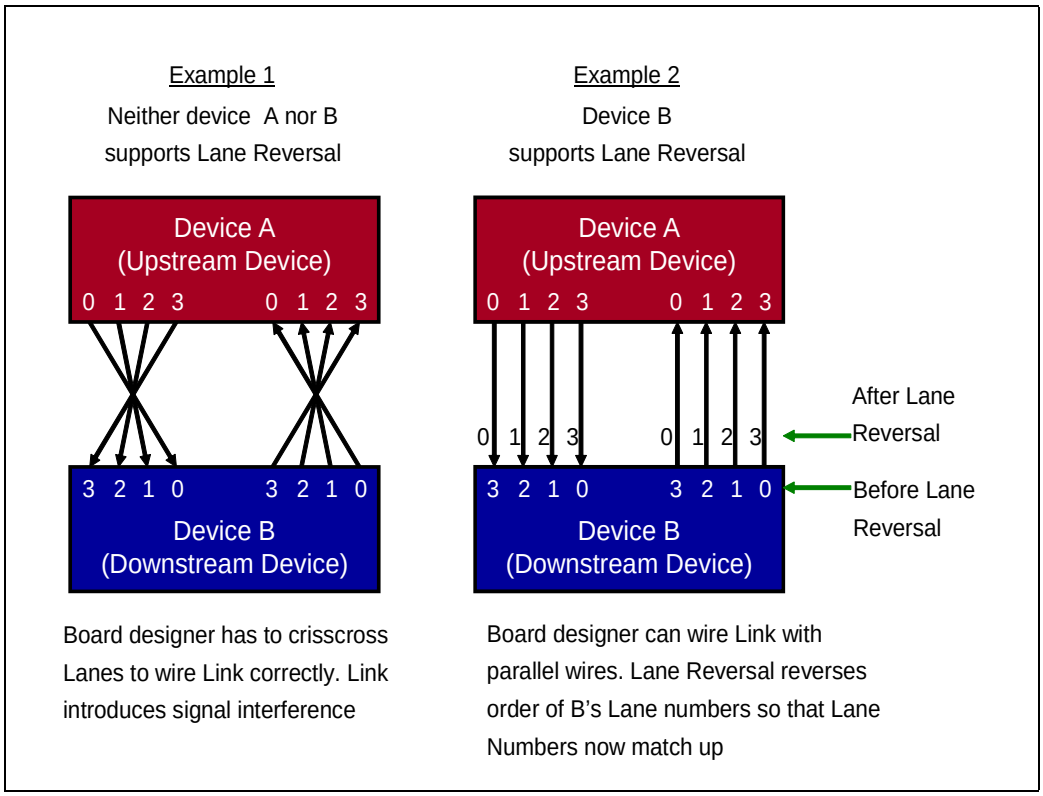
Figure 14-1: Link Training and Status State Machine Location



Due to the way the Lanes are organized on the pins of the device's package, it may not be possible to match up the Lanes of the two devices without crisscrossing the wires (see Figure 14-2 on page 502). Crisscrossed wires will introduce interference into the Link. If however, one or both of the devices support Lane Reversal, the designer could wire the Lanes in parallel fashion. During the Link training and initialization process, one device reverses the Lane numbering so the Lane numbers of the two ports would match up (Figure 14-2 on page 502). Unfortunately, the specification does not require devices to support the Lane Reversal feature. Hence, the

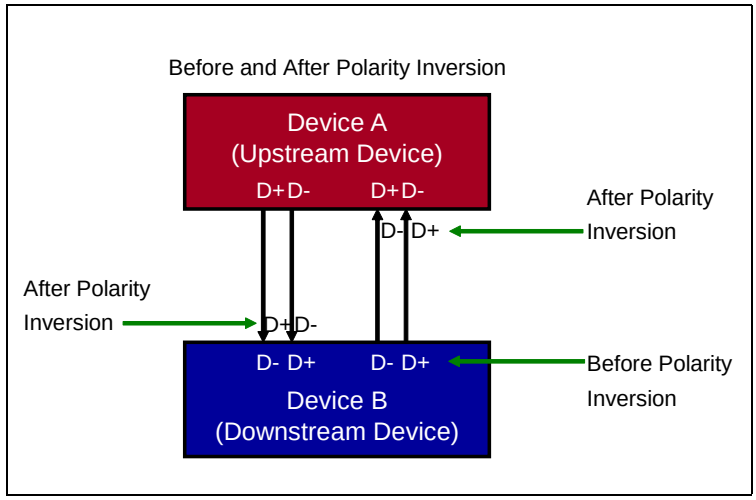
designer must verify that at least one of the two devices connected via a Link supports this feature before wiring the Lanes of the two ports in reverse order. If the device supports this feature, the Lane Reversal process may permit a multi-Lane Link to be split into multiple Links that connect to multiple devices. More on this feature later.

Figure 14-2: Example Showing Lane Reversal



- **Polarity Inversion** may be necessary. The D+ and D- differential pair terminals for two devices may not be connected correctly, or may be intentionally reversed so that the signals do not crisscross when wiring the Link. If Lanes are wired with D+ and D- of one device wired to D- and D+ of the remote device, respectively, the Polarity Inversion feature reverses the D+ and D- signal polarities of the receiver differential terminal. Figure 14-3 illustrates the benefit of this feature on a x1 Link. Support for Polarity Inversion is mandatory.

Figure 14-3: Example Showing Polarity Inversion



- **Link Data Rate.** Link initialization and training is completed at the default 2.5Gbit/s Generation 1 data rate. In the future, Generation 2 PCI Express will support higher data rates of 5Gbit/s and 10Gbit/s. During training, each node advertises its highest data rate capability. The Link is then initialized with the highest common frequency that both neighbors can support.
- **Bit Lock.** Before Link training begins, the receiver PLL is not yet sync'd with the remote transmitter's transmit clock, and the receiver is unable to differentiate between one received bit and another. During Link training, the receiver PLL is sync'd to the transmit clock and the receiver is then able to shift in the received serial bit stream. See "Achieving Bit Lock" on page 440.
- **Symbol Lock.** Before training, the receiver has no way of discerning the boundary between two, 10-bit sympols. During training, when TS1 and TS2 Ordered-Sets are exchanged, the receiver is able to locate the COM symbol (using its unique encoding) and uses it to initialize the deserializer. See "Symbol Boundary Sensing (Symbol Lock)" on page 441.
- **Lane-to-Lane De-skew**. Due to Link wire length variations and the different driver/receiver characteristics on a multi-Lane Link, each of the parallel bit streams that represent a packet are transmitted simultaneously, but they do not arrive at the receivers on each lane at the same time. The receiver circuit must compensate for this skew by adding or removing delays on each Lane so that the receiver can receive and align the serial bit streams of the packet (see "Lane-to-Lane De-Skew" on page 444). This deskew feature combined with the Polarity Inversion and Lane Reversal features, greatly simplifies

the designer's task of wiring up the high speed Link.

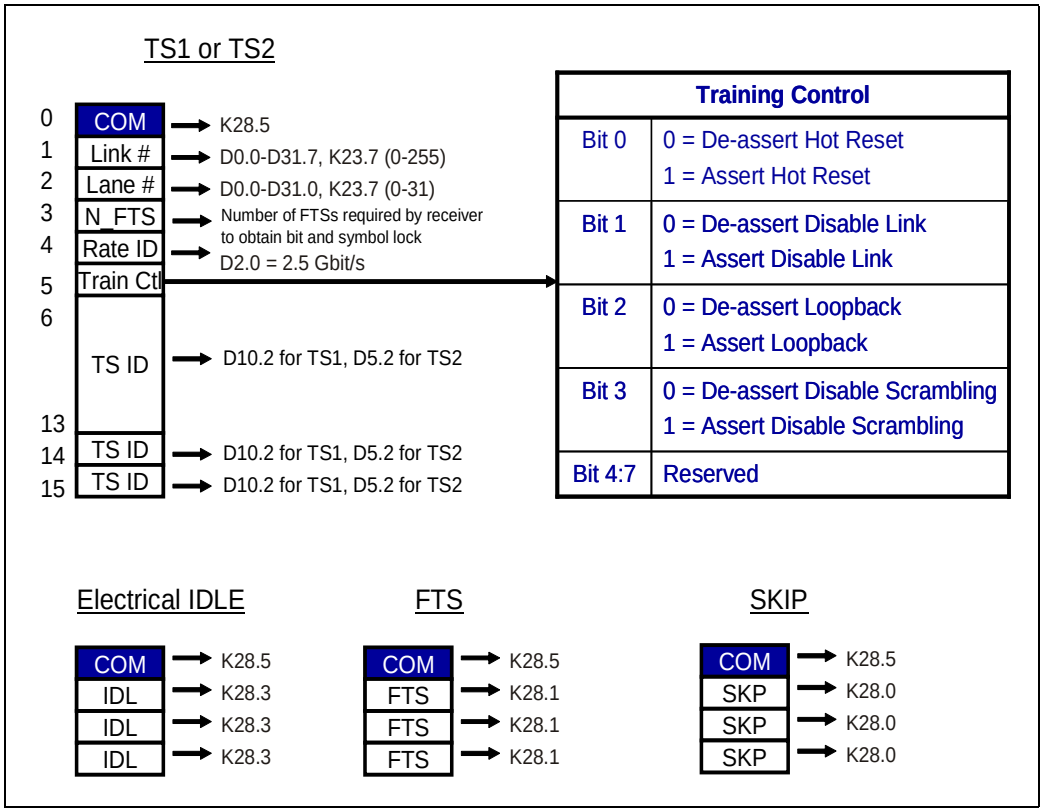
Ordered-Sets Used During Link Training and Initialization

Physical Layer Packets (PLPs), referred to as Ordered-Sets, are exchanged between neighboring devices during the Link training and initialization process. These packets were briefly described in the section on "Ordered-Sets" on page 433. The five Ordered-Sets are:

- Training Sequence 1 and 2 (TS1 and TS2),
- Electrical Idle,
- Fast Training Sequence (FTS), and
- Skip (SKIP) Ordered-Sets.

Their character structure is summarized in Figure 14-4 on page 504.

Figure 14-4: Five Ordered-Sets Used in the Link Training and Initialization Process



15 Power Budgeting

The Previous Chapter

The previous chapter described the function of the Link Training and Status State Machine (LTSSM) of the Physical Layer. It also described the initialization process of the Link from Power-On or Reset, until the full-on L0 state, where traffic on the Link can begin. In addition, the chapter described the lower power management states L0s, L1, L2, L3 and briefly discusses entry and exit procedure to/from these states.

This Chapter

This chapter describes the mechanisms that software can use to determine whether the system can support an add-in card based on the amount of power and cooling capacity it requires.

The Next Chapter

The next chapter provides a detailed description of PCI Express power management, which is compatible with revision 1.1 of the *PCI Bus PM Interface Specification* and the *Advanced Configuration and Power Interface,* revision 2.0 (ACPI). In addition PCI Express defines extensions that are orthogonal to the PCI-PM specification. These extensions focus primarily on Link Power and PM event management. This chapter also provides an overall context for the discussion of power management, by including a description of the OnNow Initiative, ACPI, and the involvement of the Windows OS is also provided.

Introduction to Power Budgeting

The primary goal of the PCI Express power budgeting capability is to allocate power for PCI Express hot plug devices, which can be added to the system during runtime. This capability ensures that the system can allocate the proper amount of power and cooling for these devices.

The specification states that “power budgeting capability is optional for PCI Express devices implemented in a form factor which does not require hot plug, or that are integrated on the system board.” None of the form factor specifications released at the time of this writing required support for hot plug and did not require the power budgeting capability. However, form factor specifications under development will require hot plug support and may also require power budgeting capability.

System power budgeting is always required to support all system board devices and add-in cards. The new power budgeting capability provides mechanisms for managing the budgeting process. Each form factor specification defines the minimum and maximum power for a given expansion slot. For example, the Electromechanical specification limits the amount of power an expansion card can consume prior to and during configuration, but after a card is configured and enabled, it can consume the maximum amount of power specified for the slot. Chapter 18, entitled "Add-in Cards and Connectors," on page 685. Consequently, in the absence of the power budgeting capability registers, the system designer is responsible for guaranteeing that power has been budgeted correctly and that sufficient cooling is available to support any compliant card installed into the connector.

The specification defines the configuration registers that are designed to support the power budgeting process, but does not define the power budgeting methods and processes. The next section describes the hardware and software elements that would be involved in power budgeting, including the specified configuration registers.

The Power Budgeting Elements

Figure 15-2 illustrates the concept of Power Budgeting for hot plug cards. The role of each element involved in the power budgeting, allocation, and reporting process is listed and described below:

- System Firmware Power Management (used during boot time)
- Power Budget Manager (used during run time)
- Expansion Ports (ports to which card slots are attached)
- Add-in Devices (Power Budget Capable)

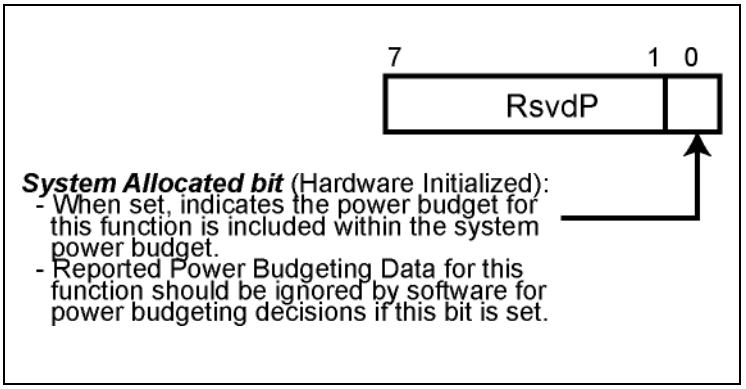
System Firmware — System firmware, having knowledge of the system design, is responsible for reporting system power information. The specification recommends the following power information be reported to the PCI Express power budget manager, which allocates and verifies power consumption and dissipa-

tion during runtime:

- Total system power available.
- Power allocated to system devices by firmware
- Number and type of slots in the system.

Firmware may also allocate power to PCI Express devices that support the power budgeting capability configuration register set (e.g., a hot-plug device used during boot time). The Power Budgeting Capability register (see Figure 15-1) contains a System Allocated bit that is intended to be set by firmware to notify the power budget manager that power for this device has been included in the system power allocation. Note that the power manager must read and save power information for hot-plug devices that are allocated by the system, in case they are removed during runtime.

Figure 15-1: System Allocated Bit



The Power Manager — The power manager initializes when the OS installs, at which time it receives power-budget information from system firmware. The specification does not define the method for communicating this information.

The power budget manager is responsible for allocating power for all PCI Express devices. This allocation includes:

- PCI Express devices that have not already been allocated by the system (includes embedded devices that support power budgeting).
- Hot-plugged devices installed at boot time.
- New devices added during runtime.

Expansion Ports — Figure 15-2 on page 561 illustrates a hot plug port that must have the Slot Power Limit and Slot Power Scale fields within the Slot Capabilities register implemented. The firmware or power budget manager must load these fields with a value that represents the maximum amount of power supported by this port. When software writes to these fields the port delivers the Set_Slot_Power_Limit message to the device. These fields are also written when software configures a card that has been added during a hot plug installation.

The PCI Express specification requires that:

- Any downstream port of a Switch or a Root Complex that has a slot attached (i.e., the Slot Implemented bit within its PCI Express Capabilities register is set) must implement the Slot Capabilities register.
- Software must initialize the Slot Power Limit Value and Scale fields of the Slot Capabilities register of the Switch or Root Complex Downstream Port that is connected to an add-in slot.
- The Upstream Port of an Endpoint, Switch, or a PCI Express-PCI Bridge must implement the Device Capabilities register.
- When a card is installed in a slot, and software updates the power limit and scale values in the Downstream port of the Switch or Root Complex, that port will automatically transmit the Set_Slot_Power_Limit message to the Upstream Port of the Endpoint, Switch, or a PCI Express-PCI Bridge on the installed card.
- The recipient of the Message must use the value in the Message data payload to limit usage of the power for the entire card/module, unless the card/module will never exceed the lowest value specified in the corresponding electromechanical specification.

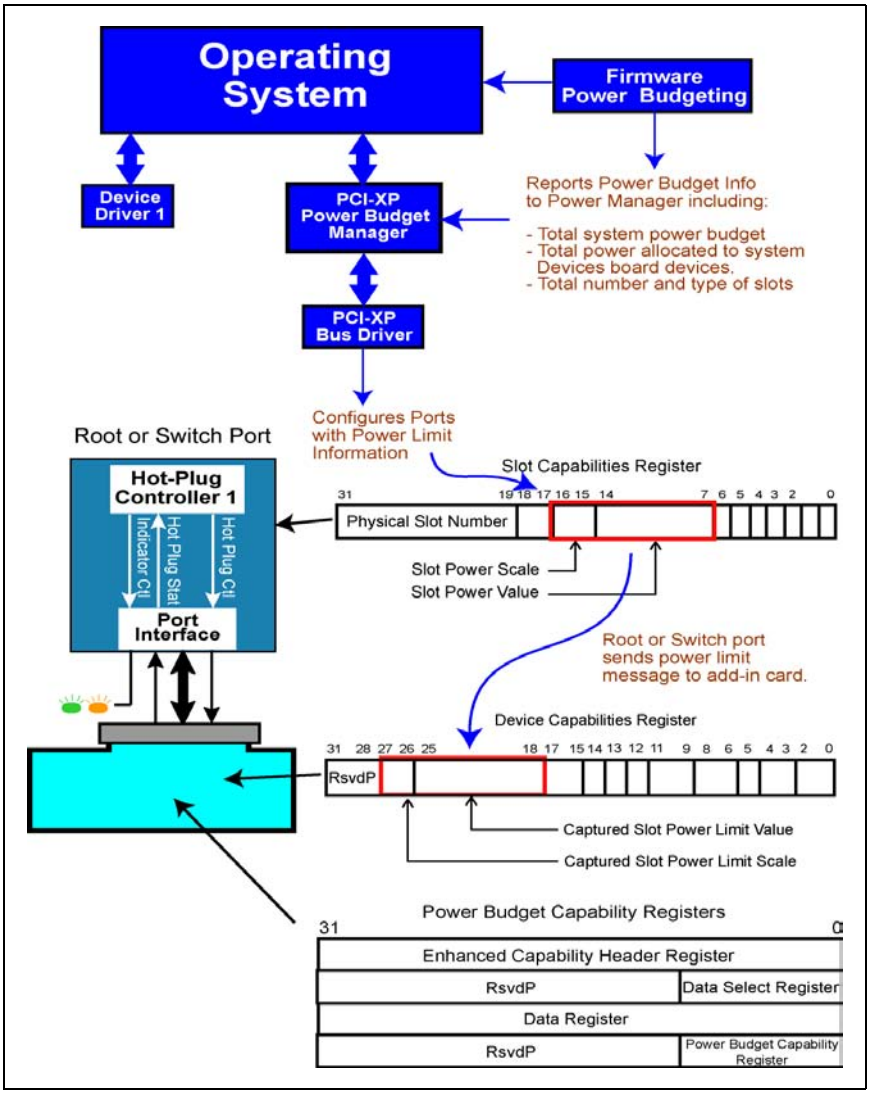
Add-in Devices—Expansion cards that support the power budgeting capability must include the:

- Slot Power Limit Value and Slot Limit Scale fields within the Device Capabilities register.
- Power Budgeting Capability register set for reporting power-related information.

These devices must not consume more power than the lowest power specified by the form factor specification. Once power budgeting software allocates additional power via the Set_Slot_Power_Limit message, the device can consume the power specified, but not until it has been configured and enabled.

Device Driver—The device's software driver is responsible for verifying that sufficient power is available for proper device operation prior to enabling it. If the power is lower than that required by the device, the device driver is responsible for reporting this to a higher software authority.

Figure 15-2: Elements Involved in Power Budget



Slot Power Limit Control

Software is responsible for determining the maximum amount of power that an expansion device is allowed to consume. This power allocation is based on the power partitioning within the system, thermal capabilities, etc. Knowledge of the system's power and thermal limits comes from system firmware. The firmware or power manager (which receives power information from firmware) is responsible for reporting the power limits to each expansion port.

Expansion Port Delivers Slot Power Limit

Software writes to the *Slot Power Limit Value* and *Slot Power Limit Scale* fields of the Slot Capability register to specify the maximum power that can be consumed by the device. Software is required to specify a power value that reflects one of the maximum values defined by the specification. For example, the electromechanical specification defines maximum power listed in Table 15-1.

Table 15-1: Maximum Power Consumption for System Board Expansion Slots

	X1 Link		**X4/X8 Link**	**X16 Link**	
Standard Height	10W (max)	25W (max)	25W (max)	25W (max)	40W (max)
Low Profile Card	10W (max)		10W (max)	25W (max)	

When these registers are written by power budget software, the expansion port sends a Set_Slot_Power_Limit message to the expansion device. This procedure is illustrated in Figure 15-3 on page 563.

16 Power Management

The Previous Chapter

The previous chapter described the mechanisms that software can use to determine whether the system can support an add-in card based on the amount of power and cooling capacity it requires.

This Chapter

This chapter provides a detailed description of PCI Express power management, which is compatible with revision 1.1 of the *PCI Bus PM Interface Specification* and the *Advanced Configuration and Power Interface,* revision 2.0 (ACPI). In addition PCI Express defines extensions that are orthogonal to the PCI-PM specification. These extensions focus primarily on Link Power and PM event management. This chapter also provides an overall context for the discussion of power management, by including a description of the OnNow Initiative, ACPI, and the involvement of the Windows OS is also provided.

The Next Chapter

PCI Express includes native support for hot plug implementations. The next chapter discusses hot plug and hot removal of PCI Express devices. The specification defines a standard usage model for all device and platform form factors that support hot plug capability. The usage model defines, as an example, how push buttons and indicators (LED's) behave, if implemented on the chassis, add-in card or module. The definitions assigned to the indicators and push buttons, described in this chapter, apply to all models of hot plug implementations.

Introduction

PCI Express power management (PM) defines two major areas of support:

- **PCI-Compatible Power Management**. PCI Express power management is based upon hardware and software compatible with the *PCI Bus Power Management Interface Specification*, Revision 1.1 (also referred to as PCI-PM) and the *Advanced Configuration and Power Interface Specification*, Revision 2.0 (commonly known as ACPI). This support requires that all PCI Express functions include the PCI Power Management Capability registers, which permits transitions between function PM states.
- **Native PCI Express Extensions**. These extensions define autonomous hardware-based Link Power Management, mechanisms for waking the system, a Message transaction to report Power Management Events (PME), and low power to active state latency reporting and calculation.

This chapter is segmented into five major sections:

1. The first section is intended as a primer for the discussion of power management, by reviewing the role of system software in controlling power management features. This section restricts the discussion to the power-management software from the Windows Operating System perspective.
2. The second section "Function Power Management" on page 585 discusses PCI-PM required by PCI Express for placing functions into their low power states. This section also documents the PCI-PM capability registers used in PCI Express. Note that some of the register definitions are modified or not used by PCI Express functions.
3. Next, "Link Active State Power Management" on page 608 describes the autonomous Link power management that occurs when a device is in its active state (D0). Active State Power Management (ASPM) is a hardware-based link power conservation mechanism. Software enables ASPM and reads latency values to determine the level of ASPM appropriate, but does not initiate transitions into ASPM.
4. The third section "Software Initiated Link Power Management" on page 629 discusses the link power management, which is triggered by PCI-PM software when it changes the power state of a device. PCI Express devices are required to automatically conserve link power when software places a device into a low power state, including D3cold, (caused by the reference clock and main power being completely removed from a device).
5. Finally, "Link Wake Protocol and PME Generation" on page 638 covers Power Management Events (PME) and wakeup signaling. Devices may

request that software return them to the active state so they can handle an event that has occurred. This is done by sending PME messages. When power has been removed from a device, auxiliary power is required to monitor events and to signal Wakeup for reactivating the link. Once a device has been re-powered and the link has been re-trained the PME message can be sent.

Primer on Configuration Software

The *PCI Bus PM Interface Specification* describes how to implement the PCI PM registers that are required in PCI Express. These registers permit the OS to manage the power environment of both PCI and PCI Express functions.

Rather than immediately diving into a detailed nuts-and-bolts description of the *PCI Bus PM Interface Specification*, it's a good idea to begin by describing where it fits within the overall context of the OS and the system. Otherwise, this would just be a disconnected discussion of registers, bits, signals, etc. with no frame of reference.

Basics of PCI PM

The most popular OSs currently in use on PC-compatible machines are Windows 98/NT/2000/XP. This section provides an overview of how the OS interacts with other major software and hardware elements to manage the power usage of individual devices and the system as a whole. Table 16-1 on page 569 introduces the major elements involved in this process and provides a very basic description of how they relate to each other. It should be noted that neither the PCI Power Management spec nor the ACPI spec (Advanced Configuration and Power Interface) dictate the policies that the OS uses to manage power. It does, however, define the registers (and some data structures) that are used to control the power usage of PCI and PCI Express functions.

Table 16-1: Major Software/Hardware Elements Involved In PC PM

Element	Responsibility
OS	Directs the **overall system power management**.To accomplish this goal, the OS issues requests to the ACPI Driver, WDM (Windows Driver Model) device drivers, and to the PCI Express Bus Driver. Application programs that are power conservation-aware interact with the OS to accomplish device power management.

Table 16-1: Major Software/Hardware Elements Involved In PC PM (Continued)

Element	Responsibility
ACPI Driver	Manages configuration, power management, and thermal control of **devices embedded on the system board that do not adhere to any industry standard interface specification**. Examples could be chipset-specific registers, system board-specific registers that control power planes, etc. The PM registers within PCI Express functions (embedded or otherwise) are defined by the PCI PM spec and are therefore not managed by the ACPI driver, but rather by the PCI Express Bus Driver (see entry in this table).
WDM Device Driver	The WDM driver is a **Class driver** that can work with any device that falls within the Class of devices that it was written to control. The fact that it's not written for a specific device from a specific vendor means that it doesn't have register and bit-level knowledge of the device's interface. When it needs to issue a command to or check the status of the device, it issues a request to the **Miniport** driver supplied by the vendor of the specific device. The WDM also doesn't understand device characteristics that are peculiar to a specific bus implementation of that device type. As an example, the WDM doesn't understand a PCI Express device's configuration register set. It depends on the **PCI Express Bus Driver** to communicate with PCI Express configuration registers. When it receives requests from the OS to control the power state of its PCI Express device, it passes the request to the PCI Express Bus Driver: When a request to power down its device is received from the OS, the WDM saves the contents of its associated PCI Express function's device-specific registers (in other words, it performs a context save) and then passes the request to the PCI Express Bus Driver to change the power state of the device. Conversely, when a request to re-power the device is received from the OS, the WDM passes the request to the PCI Express Bus Driver to change the power state of the device. After the PCI Express Bus Driver has re-powered the device, the WDM then restores the context to the PCI Express function's device-specific registers.
Miniport Driver	**Supplied by the vendor of a device**, it receives requests from the WDM Class driver and converts them into the proper series of accesses to the device's register set.

Table 16-1: Major Software/Hardware Elements Involved In PC PM (Continued)

Element	Responsibility
PCI Express Bus Driver	This driver is **generic to all PCI Express-compliant devices**. It **manages their power states and configuration registers**, but does not have knowledge of a PCI Express function's device-specific register set (that knowledge is possessed by the Miniport Driver that the WDM driver uses to communicate with the device's register set). It receives requests from the device's WDM to change the state of the device's power management logic: When a request is received to power down the device, the PCI Express Bus Driver is responsible for saving the context of the function's PCI Express configuration Header registers and any New Capability registers that the device implements. Using the device's PCI Express configuration Command register, it then disables the ability of the device to act as a Requester or to respond as the target of transactions. Finally, it writes to the PCI Express function's PM registers to change its state. Conversely, when the device must be re-powered, the PCI Express Bus Driver writes to the PCI Express function's PM registers to change its state. It then restores the function's PCI Express configuration Header registers to their original state.
PCI Express PM registers within each PCI Express function's PCI Express configuration space.	**The location, format and usage of these registers is defined by the PCI Express PM spec**. The PCI Express Bus Driver understands this spec and therefore is the entity responsible for accessing a function's PM registers when requested to do so by the function's device driver (i.e., its WDM).
System Board power plane and bus clock control logic	The implementation and control of this logic is typically system board design-specific and is therefore **controlled by the ACPI Driver** (under the OS's direction).

OnNow Design Initiative Scheme Defines Overall PM

A whitepaper on Microsoft's website clearly defines the goals of the OnNow Design Initiative and the problems it addresses. The author has taken the liberty of reproducing the text verbatim from the *Goals* section of that paper.

Goals

The OnNow Design Initiative represents the overall guiding spirit behind the sought-after PC design. The following are the major goals as stated in an OnNow document:

- The PC is ready for use immediately when the user presses the On button.
- The PC is perceived to be off when not in use but is still capable of responding to wake-up events. Wake-up events might be triggered by a device receiving input such as a phone ringing, or by software that has requested the PC to wake up at some predetermined time.
- Software adjusts its behavior when the PC's power state changes. The operating system and applications work together intelligently to operate the PC to deliver effective power management in accordance with the user's current needs and expectations. For example, applications will not inadvertently keep the PC busy when it is not necessary, and instead will proactively participate in shutting down the PC to conserve energy and reduce noise.
- All devices participate in the device power management scheme, whether originally installed in the PC or added later by the user. Any new device can have its power state changed as system use dictates.

System PM States

Table 16-2 on page 572 defines the possible states of the overall system with reference to power consumption. The “Working”, “Sleep”, and “Soft Off” states are defined in the OnNow Design Initiative documents.

Table 16-2: System PM States as Defined by the OnNow Design Initiative

Power State	Description
Working	The system is completely usable and the OS is performing power management on a device-by-device basis. As an example, the modem may be powered down during periods when it isn’t being used.

17 Hot Plug

The Previous Chapter

The previous chapter provided a detailed description of PCI Express power management, which is compatible with revision 1.1 of the *PCI Bus PM Interface Specification* and the *Advanced Configuration and Power Interface,* revision 2.0 (ACPI). In addition PCI Express defines extensions that are orthogonal to the PCI-PM specification. These extensions focus primarily on Link Power and PM event management. This chapter also provides an overall context for the discussion of power management, by including a description of the OnNow Initiative, ACPI, and the involvement of the Windows OS is also provided.

This Chapter

PCI Express includes native support for hot plug implementations. This chapter discusses hot plug and hot removal of PCI Express devices. The specification defines a standard usage model for all device and platform form factors that support hot plug capability. The usage model defines, as an example, how push buttons and indicators (LED's) behave, if implemented on the chassis, add-in card or module. The definitions assigned to the indicators and push buttons, described in this chapter, apply to all models of hot plug implementations.

The Next Chapter

The next chapter provides an introduction to the PCI Express add-in card electromechanical specifications. It describes the card form factor, the connector details, and the auxiliary signals with a description of their function. Other card form factors are also briefly described.

Background

Some systems that employ the use of PCI and PCI-X require high availability or non-stop operation. For example, many customers require computer systems that experience downtimes of just a few minutes a year, or less. Clearly, manufacturers must focus on equipment reliability, and also provide a method of identifying and repairing equipment failures quickly. An important feature in supporting these goals is the Hot Plug/Hot Swap solutions that provide three important capabilities:

1. a method of replacing failed expansion cards without turning the system off
2. keeping the O/S and other services running during the repair
3. shutting down and restarting software associated with the failed device

Prior to the widespread acceptance of PCI many proprietary Hot Plug solutions were available to support this type of removal and replacement of expansion cards. However the original PCI implementation was not designed to support hot removal and insertion of cards, but a standardized solution for supporting this capability in PCI was needed. Consequently, two major approaches to hot replacement of PCI expansion devices have been developed. These approaches are:

- Hot Plug PCI Card — used in PC Server motherboard and expansion chassis implementations
- Hot Swap — used in CompactPCI systems based on a passive PCI backplane implementation.

In both solutions, control logic is implemented to isolate the card logic from the PCI bus via electronic switches. In conjunction with isolation logic, power, reset, and clock are controlled to ensure an orderly power down and power up of cards when they are removed and replaced. Also, status and power LEDs provide indications to the user that it is safe to remove or install the card.

The need to extend hot plug support to PCI Express cards is clear. Designers of PCI Express have incorporated Hot removal and replacement of cards as a "native" feature. The specification defines configuration registers, Hot Plug Messages, and procedures to support Hot Plug solutions.

Hot Plug in the PCI Express Environment

PCI Express Hot Plug is derived from the 1.0 revision of the Standard Hot Plug Controller specification (SHPC 1.0) for PCI. The goals of PCI Express Hot Plug are to:

- support the same "Standardized Usage Model" as defined by the Standard Hot Plug Controller specification. This ensures that the PCI Express hot plug is identical from the user perspective to existing implementations based on the SHPC 1.0 specification
- support the same software model implemented by existing operating systems. However, if the OS includes a SHPC 1.0 compliant driver, it will not work with PCI Express Hot Plug controllers, which have a different programming interface.

PCI Express defines the registers necessary to support the integration of a Hot Plug Controller within individual root and switch ports. Under Hot Plug software control, these Hot Plug controllers and the associated port interface within the root or switch port must control the card interface signals to ensure orderly power down and power up as cards are removed and replaced. Hot Plug controllers must:

- assert and deassert the PERST# signal to the PCI Express card connector
- remove or apply power to the card connector.
- Selectively turn on or turn off the Power and Attention Indicators associated with a specific card connector to draw the user's attention to the connector and advertise whether power is applied to the slot.
- Monitor slot events (e.g. card removal) and report these events to software via interrupts.

PCI Express Hot-Plug (like PCI) is designed as a "no surprises" Hot-Plug methodology. In other words, the user is not permitted to install or remove a PCI Express card without first notifying software. System software then prepares both the card and slot for the card's removal and replacement, and finally indicates to the end user (via visual indicators) status of the hot plug process and notification that installation or removal may be performed.

Surprise Removal Notification

PCI Express cards (unlike PCI) must implement the edge contacts with card presence detect pins (PRSNT1# and PRSNT2#) that break contact first (when the card is removed from the slot). This gives advanced notice to software of a "surprise" removal and enough time to remove power prior to the signals breaking contact.

Differences between PCI and PCI Express Hot Plug

The elements needed to support hot plug are essentially the same between PCI and PCI Express hot plug solutions. Figure 17-1 on page 653 depicts the PCI hardware and software elements required to support hot plug. PCI solutions implement a single standardized hot plug controller on the system board that permits all hot plug slots on the bus to be controlled by a single controller. Also, isolation logic is needed in the PCI environment to electrically disconnect a single card slot from the bus prior to card removal.

PCI Express Hot Plug differs from the PCI implementation due to point-to-point connections. (See Figure 17-2 on page 654) Point-to-point connections eliminate the need for isolation logic and permit the hot plug controller to be distributed to each port interface to which a connector is attached. A standardized software interface defined for each root and switch port permits a standardized software interface to control hot plug operations. Note that the programming interface for the PCI Express and PCI Hot Plug Controllers vary and require different software drivers.

Figure 17-1: PCI Hot Plug Elements

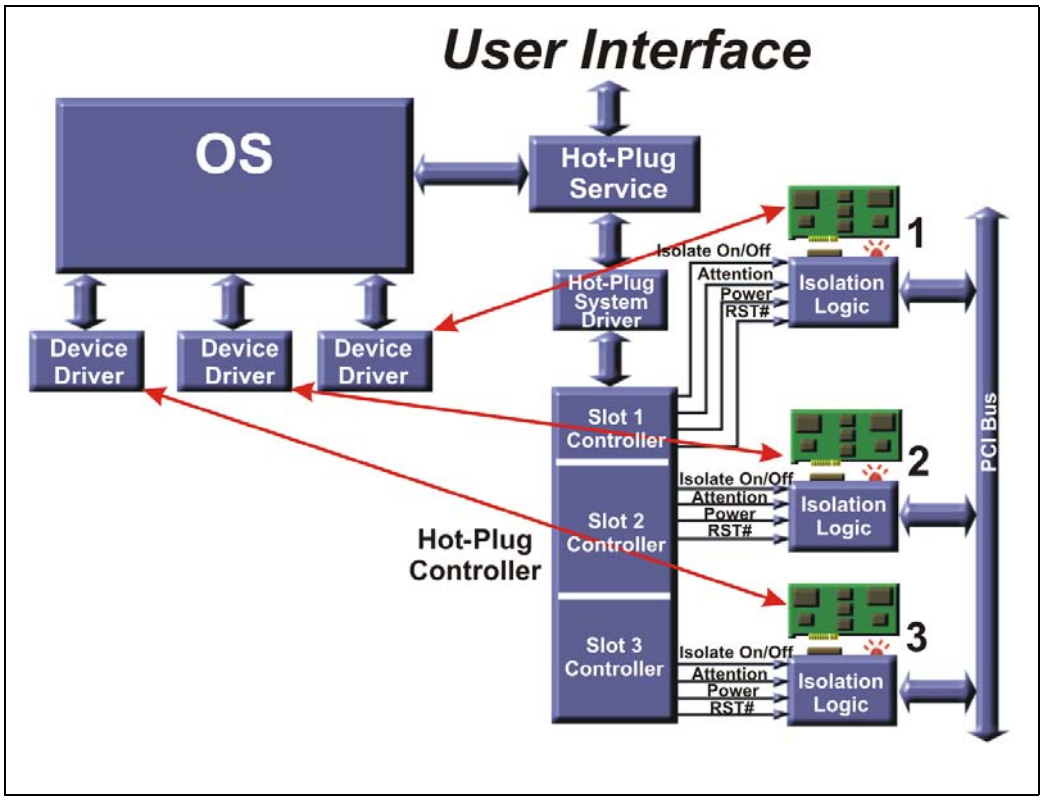
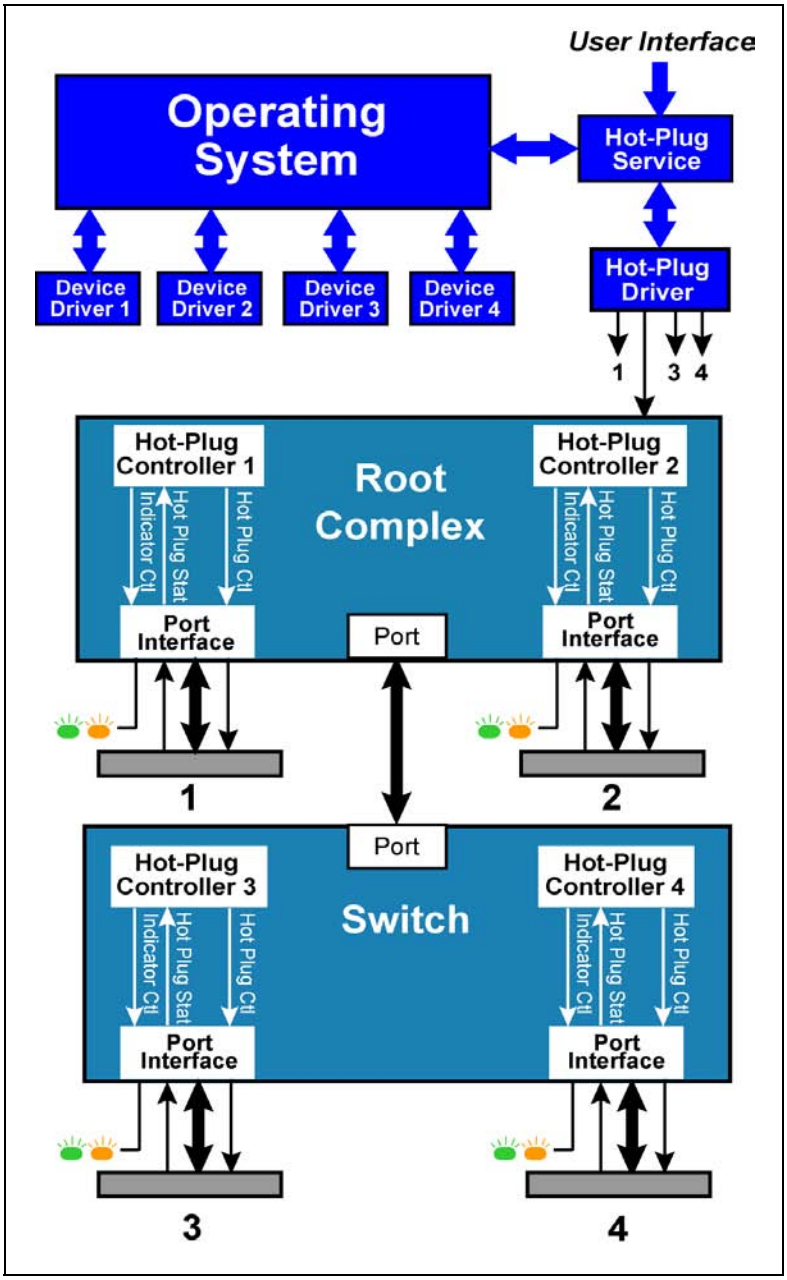


Figure 17-2: PCI Express Hot-Plug Hardware/Software Elements



18 Add-in Cards and Connectors

The Previous Chapter

PCI Express includes native support for hot plug implementations. The previous chapter discussed hot plug and hot removal of PCI Express devices. The specification defines a standard usage model for all device and platform form factors that support hot plug capability. The usage model defines, as an example, how push buttons and indicators (LED's) behave, if implemented on the chassis, add-in card or module. The definitions assigned to the indicators and push buttons, described in this chapter, apply to all models of hot plug implementations.

This Chapter

This chapter provides an introduction to the PCI Express add-in card electromechanical specifications. It describes the card form factor, the connector details, and the auxiliary signals with a description of their function. Other card form factors are also briefly described, but it should be stressed that some of them have not yet been approved by the SIG as of this writing.

The Next Chapter

The next chapter provides an introduction to configuration in the PCI Express envionment. It introduces the configuration space in which a function's configuration registers are implemented, how a function is discovered, how configuration transactions are routed, PCI-compatible space, PCI Express extended configuration space, and how to differentiate between a normal function and a bridge.

Introduction

One goal of the PCI Express add-in card electromechanical spec was to encourage migration from the PCI architecture found in many desktop and mobile devices today by making the migration path straightforward and minimizing the required hardware changes. Towards this end, PCI Express add-in cards are defined to be very similar to the current PCI add-in card form factor, allowing them to readily coexist with PCI slots in system boards designed to the ATX or micro-ATX standard. PCI Express features like automatic polarity inversion and lane reversal also help reduce layout issues on system boards, so they can still be designed using the four-layer FR4 board construction commonly used today. As a result, much of an existing system board design can remain the same when it is modified to use the new architecture, and no changes are required for existing chassis designs.

Add-in Connector

The PCI Express add-in card connector (see Figure 18-1 on page 687 and Figure 18-2 on page 688) is physically very similar to the legacy PCI connector, but uses a different pinout and does not supply -12V or 5V power. The physical dimensions of a card are the same as the PCI add-in cards and the same IO bracket is used. Table 18-1 on page 689 shows the pinout for a connector that supports PCI Express cards up to x16 (16 lanes wide). Several signals are referred to as auxiliary signals in the spec, and these are highlighted and described in more detail in the section that follows the table.

Note that cards with fewer lanes can be plugged into larger connectors that will accommodate more lanes. This is referred to as **Up-plugging**. The opposite case, installing a larger card into a smaller slot is called **Down-plugging** and, unlike PCI, is physically prevented in PCI Express by the connector keying.) Consequently, the connector described by the table will accommodate a card that is x1, x4, x8, or x16. This flexibility in the connector is highlighted by notes in the table that indicate each group of signals. For example, a x4 card plugged into this slot would only make use of pins 1 through 32, and so the note indicating the end of the x4 group of signals appears after pin 32. These segment indicators do not represent physical spaces or keys, however, because there is only one mechanical key on the connector, located between pins 11 and 12.

Figure 18-1: PCI Express x1 connector

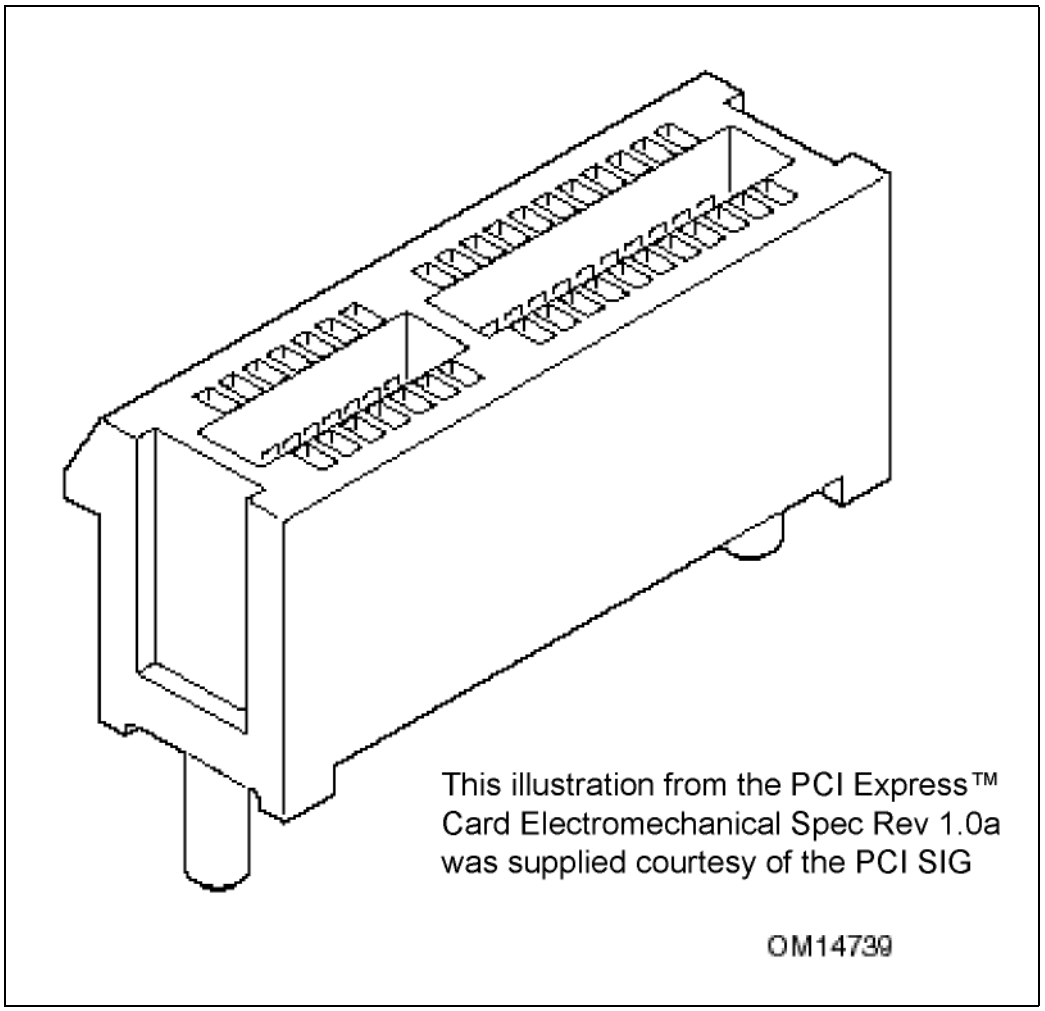


Figure 18-2: PCI Express Connectors on System Board

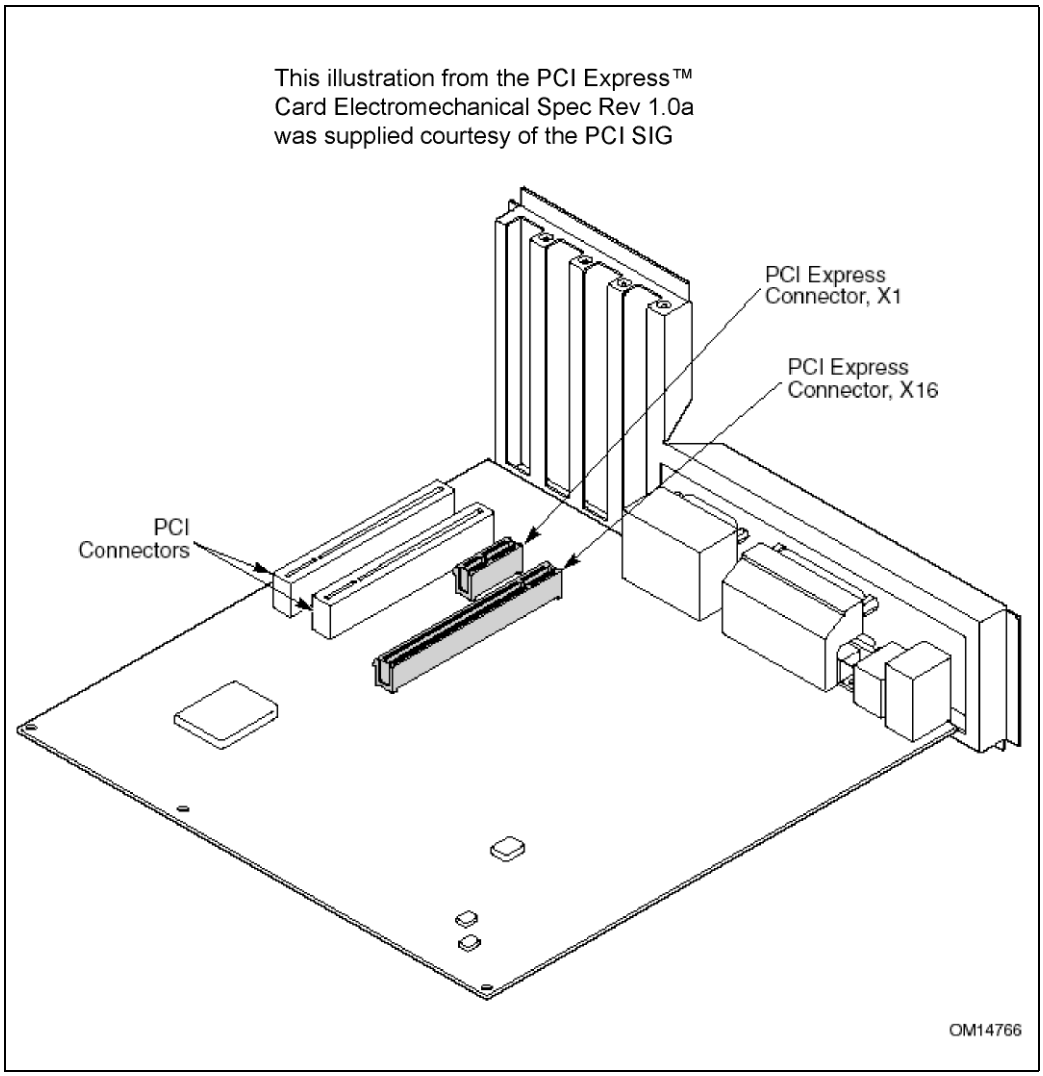


Table 18-1: PCI Express Connector Pinout

Pin #	Side B		Side A	
	Name	**Description**	**Name**	**Description**
1	+12V	12V Power	PRSNT1#	Hot-Plug presence detect
2	+12V	12V Power	+12V	12V Power
3	RSVD	Reserved	+12V	12V Power
4	GND	Ground	GND	Ground
5	SMCLK	SMBus (System Management Bus) Clock	JTAG2	TCK (Test Clock), clock input for JTAG interface
6	SMDAT	SMBus (System Management Bus) data	JTAG3	TDI (Test Data Input)
7	GND	Ground	JTAG4	TDO (Test Data output)
8	+3.3V	3.3 V Power	JTAG5	TMS (Test Mode Select)
9	JTAG1	TRST# (Test Reset) resets the JTAG interface	+3.3V	3.3 V Power
10	$3.3V_{AUX}$	3.3 V Auxiliary Power	+3.3V	3.3 V Power
11	WAKE#	Signal for link reactivation	PERST#	Fundamental reset
Mechanical Key				
12	RSVD	Reserved	GND	Ground
13	GND	Ground	REFCLK+	Reference Clock (differential pair)
14	PETp0	Transmitter differential pair, Lane 0	REFCLK-	
15	PETn0		GND	Ground

Table 18-1: PCI Express Connector Pinout (Continued)

Pin #	Side B		Side A	
	Name	**Description**	**Name**	**Description**
16	GND	Ground	PERp0	Receiver differential pair, Lane 0
17	PRSNT2#	Hot-Plug presence detect	PERn0	
18	GND	Ground	GND	Ground
End of the x1 connector				
19	PETp1	Transmitter differential pair, Lane 1	RSVD	Reserved
20	PETn1		GND	Ground
21	GND	Ground	PERp1	Receiver differential pair, Lane 1
22	GND	Ground	PERn1	
23	PETp2	Transmitter differential pair, Lane 2	GND	Ground
24	PETn2		GND	Ground
25	GND	Ground	PERp2	Receiver differential pair, Lane 2
26	GND	Ground	PERn2	
27	PETp3	Transmitter differential pair, Lane 3	GND	Ground
28	PETn3		GND	Ground
29	GND	Ground	PERp3	Receiver differential pair, Lane 3
30	RSVD	Reserved	PERn3	
31	PRSNT2#	Hot-Plug presence detect	GND	Ground
32	GND	Ground	RSVD	Reserved
End of the x4 connector				
33	PETp4	Transmitter differential pair, Lane 4	RSVD	Reserved
34	PETn4		GND	Ground

19 Configuration Overview

The Previous Chapter

The previous chapter provided an introduction to the PCI Express add-in card electromechanical specifications. It described the card form factor, the connector details, and the auxiliary signals with a description of their function. Other card form factors were also briefly described, but it should be stressed that some of them have not yet been approved by the SIG as of this writing.

This Chapter

This chapter provides an introduction to configuration in the PCI Express environment. It introduces the configuration space in which a function's configuration registers are implemented, how a function is discovered, how configuration transactions are routed, PCI-compatible space, PCI Express extended configuration space, how a function is discovered, and how to differentiate between a normal function and a bridge.

The Next Chapter

The next chapter provides a detailed description of the two configuration mechanisms used in a PCI Express platform: the PCI-compatible configuration mechanism, and the PCI Express enhanced configuration mechanism. It provides a detailed description of the initialization period immediately following power-up, as well as error handling during this period.

Definition of Device and Function

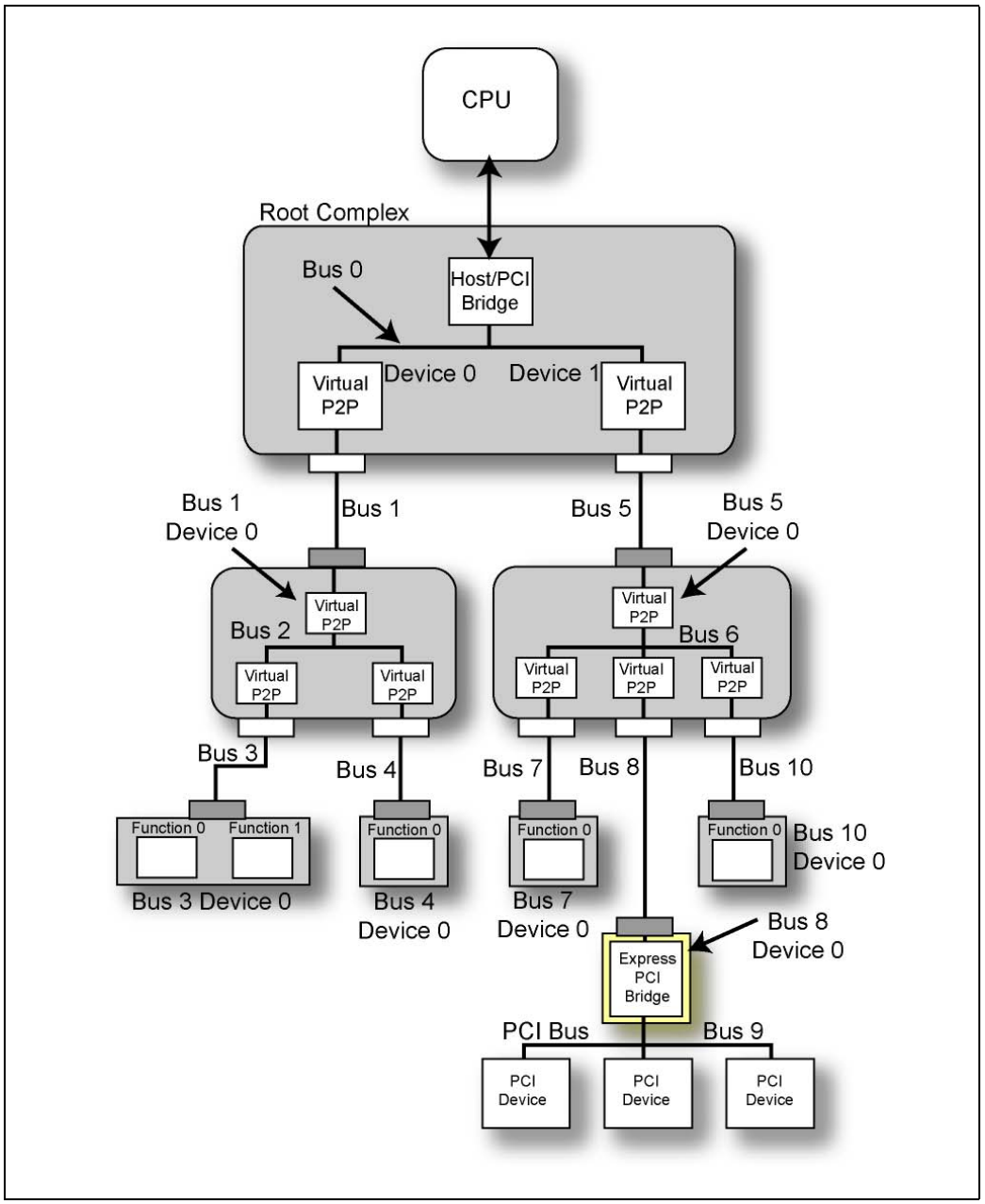
Just as in the PCI environment, a device resides on a bus and contains one or more functions (a device containing multiple functions is referred to as a multifunction device). Each of the functions within a multifunction device provides a stand-alone functionality. As an example, one function could be a graphics controller while another might be a network interface.

Just as in PCI, a device may contain up to a maximum of eight functions numbered 0-through-7:

- The one-and-only function implemented in a single-function device must be function 0.
- In a multifunction device, the first function must be function 0, while the remaining functions do not have to be implemented in a sequential manner. In other words, a device could implement functions 0, 2, and 7.

In Figure 19-1 on page 713, Device 0 on Bus 3 is a multifunction device containing two functions, each of which implements its own set of configuration registers.

Figure 19-1: Example System



Definition of Primary and Secondary Bus

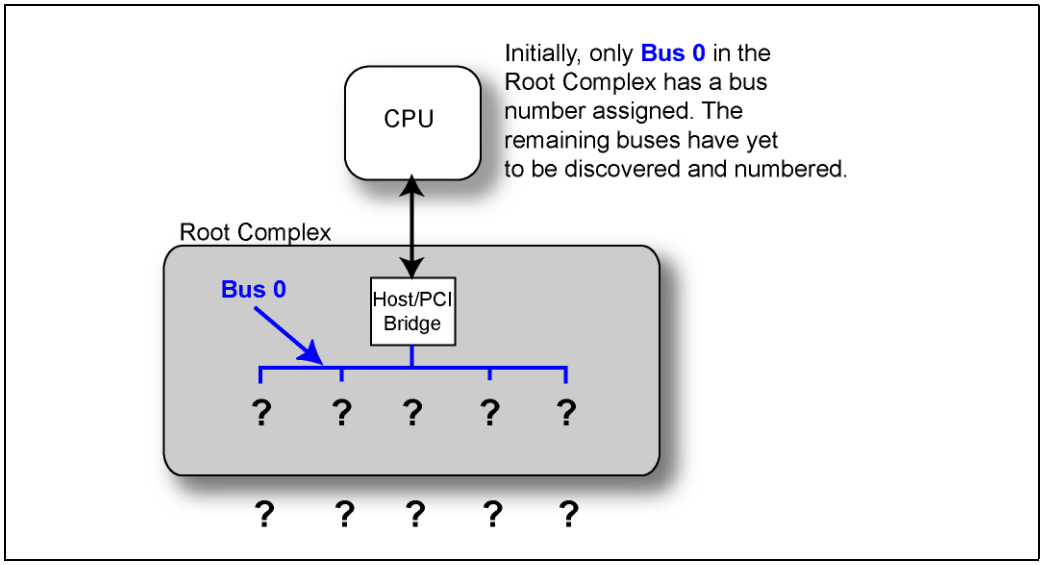
The bus connected to the upstream side of a bridge is referred to as its primary bus, while the bus connected to its downstream side is referred to as its secondary bus.

Topology Is Unknown At Startup

Refer to Figure 19-2 on page 714. When the system is first powered up, the configuration software has not yet scanned the PCI Express fabric to discover the machine topology and how the fabric is populated. The configuration software is only aware of the existence of the Host/PCI bridge within the Root Complex and that bus number 0 is directly connected to the downstream (i.e., secondary) side of the bridge.

It has not yet scanned bus 0 and therefore does not yet know how many PCI Express ports are implemented on the Root Complex. The process of scanning the PCI Express fabric to discover its topology is referred to as the *enumeration* process.

Figure 19-2: Topology View At Startup



Each Function Implements a Set of Configuration Registers

Introduction

At the behest of software executing on the processor, the Root Complex initiates configuration transactions to read from or write to a function's configuration registers. These registers are accessed to discover the existence of a function as well as to configure it for normal operation. In addition to memory, IO, and message space, PCI Express also defines a dedicated block of configuration space allocated to each function within which its configuration registers are implemented.

Function Configuration Space

Refer to Figure 19-3 on page 717. Each function's configuration space is 4KB in size and is populated as described in the following two subsections.

PCI-Compatible Space

The 256 byte (64 dword) PCI-compatible space occupies the first 256 bytes of this 4KB space. It contains the function's PCI-compatible configuration registers. This area can be accessed using either of two mechanisms (both of which are described later):

- The PCI configuration access mechanism (see "PCI-Compatible Configuration Mechanism" on page 723).
- The PCI Express Enhanced Configuration mechanism (see "PCI Express Enhanced Configuration Mechanism" on page 731).

The first 16 dwords comprises the PCI configuration header area, while the remaining 48 dword area is reserved for the implementation of function-specific configuration registers as well as PCI New Capability register sets. It is mandatory that each PCI Express function must implement the PCI Express Capability Structure (defined later) within this area. A full description of the PCI-compatible registers may be found in "PCI Compatible Configuration Registers" on page 769.

PCI Express Extended Configuration Space

The remaining 3840 byte (960 dword) area is referred to as the PCI Express Extended Configuration Space. It is utilized to implement the optional PCI Express Extended Capability registers:

- Advanced Error Reporting Capability register set.
- Virtual Channel Capability register set.
- Device Serial Number Capability register set.
- Power Budgeting Capability register set.

A full description of the these optional register sets may be found in "Express-Specific Configuration Registers" on page 893.

Host/PCI Bridge's Configuration Registers

The Host/PCI bridge's configuration register set does not have to be accessed using either of the spec-defined configuration mechanisms mentioned in the previous section. Rather, it is mapped into a Root Complex design-specific address space (almost certainly memory space) that is known to the platform-specific BIOS firmware. However, its configuration register layout and usage must adhere to the standard Type 0 template defined by the PCI 2.3 spec (see "Header Type 0" on page 770 for details on the Type 0 register template).

20 Configuration Mechanisms

The Previous Chapter

The previous chapter provided an introduction to configuration in the PCI Express environment. It introduced the configuration space in which a function's configuration registers are implemented, how a function is discovered, how configuration transactions are routed, PCI-compatible space, PCI Express extended configuration space, and how to differentiate between a normal function and a bridge.

This Chapter

This chapter provides a detailed description of the two configuration mechanisms used in a PCI Express platform: the PCI-compatible configuration mechanism, and the PCI Express enhanced configuration mechanism. It provides a detailed description of the initialization period immediately following power-up, as well as error handling during this period.

The Next Chapter

The next chapter provides a detailed description of the discovery process and bus numbering. It describes:

- Enumerating a system with a single Root Complex
- Enumerating a system with multiple Root Complexes
- A multifunction device within a Root Complex or a Switch
- An Endpoint embedded in a Switch or Root Complex
- Automatic Requester ID assignment.
- Root Complex Register Blocks (RCRBs)

Introduction

Refer to Figure 20-1 on page 723. Each function implements a 4KB configuration space. The lower 256 bytes (64 dwords) is the PCI-compatible configuration space, while the upper 960 dwords is the PCI Express extended configuration space.

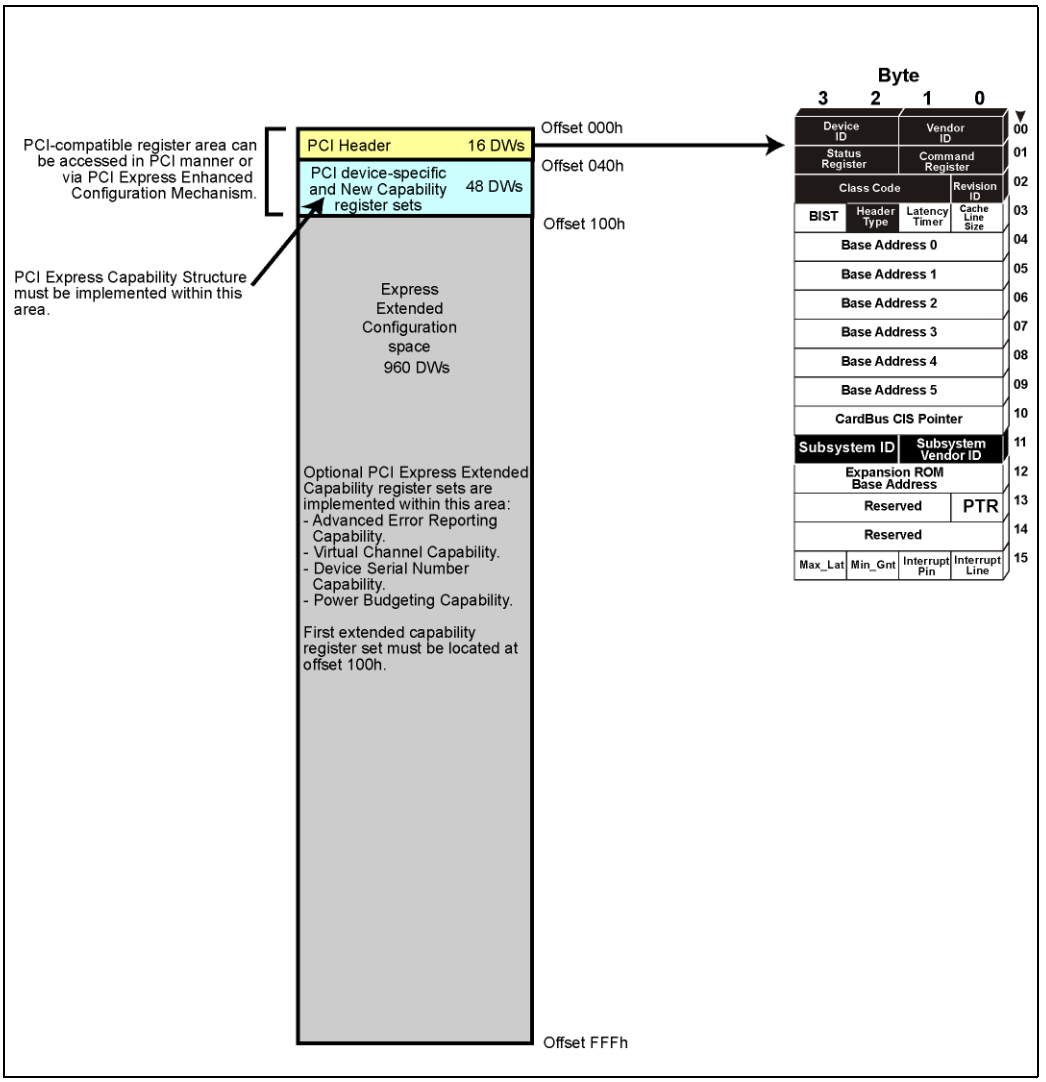
There are two mechanisms available that allow configuration software running on the processor to stimulate the Root Complex to generate configuration transactions:

- The PCI 2.3-compatible configuration access mechanism.
- The PCI express enhanced configuration mechanism.

These two mechanisms are described in this chapter.

Intel x86 and PowerPC processors (as two example processor families) do not possess the ability to perform configuration read and write transactions. They use memory and IO (IO is only in the x86 case) read and write transactions to communicate with external devices. This means that the Root Complex must be designed to recognize certain IO or memory accesses initiated by the processor as requests to perform configuration accesses.

Figure 20-1: A Function's Configuration Space



PCI-Compatible Configuration Mechanism

For x86-based PC-AT compatible systems, the 2.3 PCI spec defines a method that utilizes processor-initiated IO accesses to instruct the host/PCI bridge (in this case, within the Root Complex) to perform PCI configuration accesses.The

spec does not define a configuration mechanism to be used in systems other than PC-AT compatible systems.

Background

The x86 processor family is capable of addressing up to, but no more than, 64KB of IO address space. In the EISA spec, the usage of this IO space was defined in such a manner that the only IO address ranges available for the implementation of the PCI Configuration Mechanism (without conflicting with an ISA or EISA device) were 0400h - 04FFh, 0800h - 08FFh, and 0C00h - 0CFFh. Many EISA system board controllers already resided within the 0400h - 04FFh address range, making it unavailable.

Consider the following:

- As with any other PCI function, a host/PCI bridge may implement up to 64 dwords of configuration registers.
- Each PCI function on each PCI bus requires 64 dwords of dedicated configuration space.

Due to the lack of available IO real estate within the 64KB of IO space, it wasn't feasible to map each configuration register directly into the processor's IO address space. Alternatively, the system designer could implement the configuration registers within the processor's memory space. The amount of memory space consumed aside, the address range utilized would be unavailable for allocation to regular memory. This would limit the system's flexibility regarding the mapping of actual memory.

PCI-Compatible Configuration Mechanism Description

General

The PCI-Compatible Configuration Mechanism utilizes two 32-bit IO ports implemented in the Host/PCI bridge within the Root Complex, located at IO addresses 0CF8h and 0CFCh. These two ports are:

- The 32-bit **Configuration Address Port**, occupying IO addresses 0CF8h through 0CFBh.
- The 32-bit **Configuration Data Port**, occupying IO addresses 0CFCh through 0CFFh.

Accessing one of a function's PCI-compatible configuration registers is a two step process:

1. Write the target bus number, device number, function number and dword number to the Configuration Address Port and set the Enable bit in it to one.
2. Perform a one-byte, two-byte, or four-byte IO read from or a write to the Configuration Data Port.

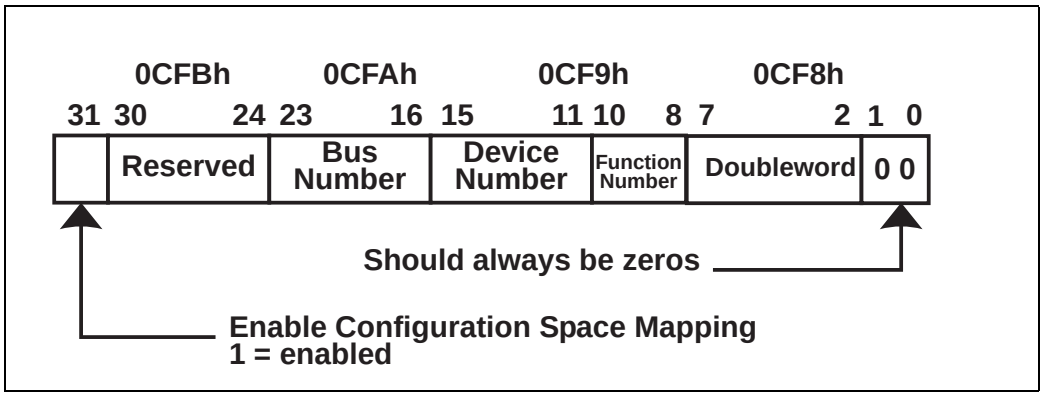
In response, the host/PCI bridge within the Root Complex compares the specified target bus to the range of buses that exist on the other side of the bridge and, if the target bus resides beyond the bridge, it initiates a configuration read or write transaction (based on whether the processor is performing an IO read or write with the Configuration Data Port).

Configuration Address Port

Refer to Figure 20-2 on page 726. The Configuration Address Port only latches information when the processor performs a full 32-bit write to the port. A 32-bit read from the port returns its contents. The assertion of reset clears the port to all zeros. Any 8- or 16-bit access within this IO dword is treated as an 8- or 16-bit IO access. The 32-bits of information written to the Configuration Address Port must conform to the following template (illustrated in Figure 20-2 on page 726):

- bits **[1:0]** are hard-wired, read-only and must return **zeros** when read.
- bits **[7:2]** identify the **target dword** (1-of-64) within the target function's PCI-compatible configuration space. When the Root Complex subsequently generates the resultant configuration request packet, this bit field supplies the content of the packet's Register Number field and the packet's Extended Register Number field is set to all zeros. This configuration access mechanism is therefore limited to addressing the first 64 dwords of the targeted function's configuration space (i.e., the function's PCI-compatible address space).
- bits **[10:8]** identify the **target function** number (1-of-8) within the target device.
- bits **[15:11]** identify the target device number (1-of-32).
- bits **[23:16]** identifies the **target bus** number (1-of-256).
- bits **[30:24]** are **reserved** and must be zero.
- bit **31 must be** set to a **one**, enabling the translation of a subsequent processor IO access to the Configuration Data Port into a configuration access. If bit 31 is zero and the processor initiates an IO read from or IO write to the Configuration Data Port, the transaction is treated as an IO transaction request.

Figure 20-2: Configuration Address Port at 0CF8h



Bus Compare and Data Port Usage

Refer to Figure 20-3 on page 728. The Host/PCI bridge within the Root Complex implements a Bus Number register and a Subordinate Bus Number register. In a chipset that only supports one Root Complex, the bridge may have a bus number register that is hardwired to 0, a read/write register that reset forces to 0, or it just implicitly knows that it is the bridge to bus 0. If bit 31 in the Configuration Address Port (see Figure 20-2 on page 726) is enabled (i.e., set to one), the bridge compares the target bus number to the range of buses that exists beyond the bridge.

Target Bus = 0. If the target bus is the same as the value in the Bus Number register, this is a request to perform a configuration transaction on bus 0. A subsequent IO read from or write to the bridge's Configuration Data Port at 0CFCh causes the bridge to generate a Type 0 configuration read or write transaction. When devices that reside on a PCI bus detect a Type 0 configuration transaction in progress, this informs them that one of them is the target device (rather than a device on one of the subordinate buses beneath the bus the Type 0 transaction is being performed on).

21 PCI Express Enumeration

The Previous Chapter

The previous chapter provided a detailed description of the two configuration mechanisms used in a PCI Express platform: the PCI-compatible configuration mechanism, and the PCI Express enhanced configuration mechanism. It provided a detailed description of the initialization period immediately following power-up, as well as error handling during this period.

This Chapter

This chapter provides a detailed description of the discovery process and bus numbering. It describes:

- Enumerating a system with a single Root Complex
- Enumerating a system with multiple Root Complexes
- A multifunction device within a Root Complex or a Switch
- An Endpoint embedded in a Switch or Root Complex
- Automatic Requester ID assignment.
- Root Complex Register Blocks (RCRBs)

The Next Chapter

The next chapter provides a detailed description of the configuration registers residing a function's PCI-compatible configuration space. This includes the registers for both non-bridge and bridge functions.

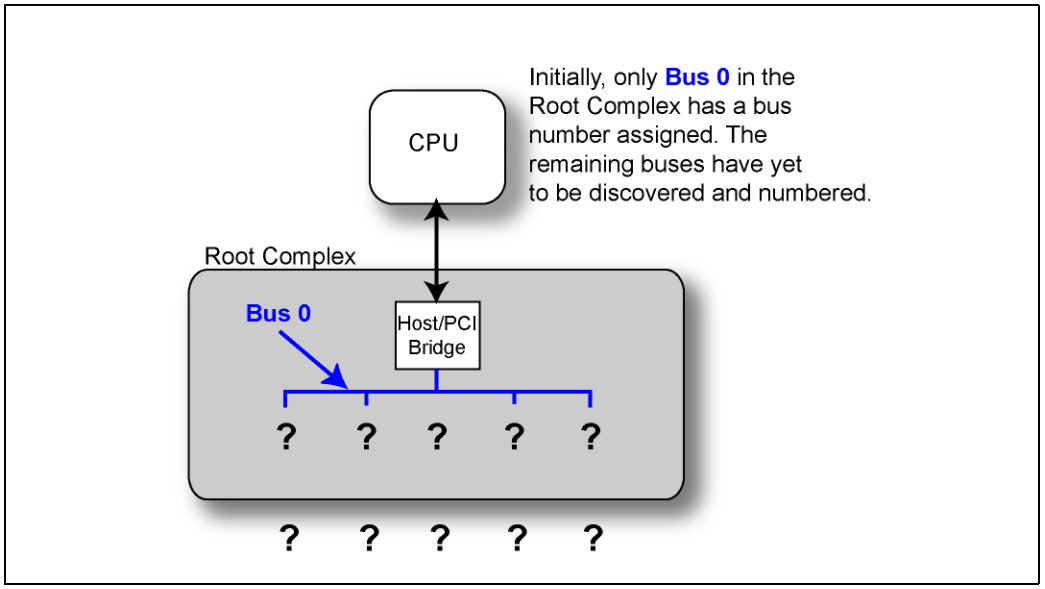
Introduction

The discussions associated with Figure 19-1 on page 713 and Figure 20-4 on page 730 assumed that, each of the buses had been discovered and numbered earlier in time.

In reality, at power up time, the configuration software only knows of the existence of bus 0 (the bus that resides on the downstream side of the Host/PCI bridge) and does not even know what devices reside on bus 0 (see Figure 21-1 on page 742).

This chapter describes the enumeration process: the process of discovering the various buses that exist and the devices and functions which reside on each of them.

Figure 21-1: Topology View At Startup



Enumerating a System With a Single Root Complex

Figure 21-2 on page 748 illustrates an example system before the buses and devices have been enumerated, while Figure 21-3 on page 749 shows the same system after the buses and devices have been enumerated. The discussion that follows assumes that the configuration software uses either of the two configuration mechanisms defined in the previous chapter. At startup time, the configuration software executing on the processor performs bus/device/function enumeration in the following manner:

1. Starting with device 0 (bridge A), the enumeration software attempts to read the Vendor ID from function 0 in each of the 32 possible devices on bus 0.
   - If a valid (not FFFFh) Vendor ID is returned from bus 0, device 0, function 0, this indicates that the device is implemented and contains at least one function. Proceed to the next step.
   - If a value of FFFFh were returned as the Vendor ID, this would indicate that function 0 is not implemented in device 0. Since it is a rule that the first function implemented in any device must be function 0, this would mean that device was not implemented and the enumeration software would proceed to probe bus 0, device 1, function 0.
2. The Header Type field (see Figure 21-6 and Figure 21-7) in the Header register (see Figure 21-4) contains the value one (0000001b) indicating that this is a PCI-to-PCI bridge with the PCI-compatible register layout shown in Figure 21-7 on page 752. This discussion assumes that the Multifunction bit (bit 7) in the Header Type register is 0, indicating that function 0 is the only function in this bridge. *It should be noted that the spec does not preclude implementing multiple functions within this bridge and each of these functions, in turn, could represent virtual PCI-to-PCI bridges.*
3. Software now performs a series of configuration writes to set the bridge's bus number registers as follows:
   - Primary Bus Number Register = 0.
   - Secondary Bus Number Register = 1.
   - Subordinate Bus Number Register = 1.

   The bridge is now aware that the number of the bus directly attached to its downstream side is 1 (Secondary Bus Number = 1) and the number of the bus farthest downstream of it is 1 (Subordinate Bus Number = 1).
4. Software updates the Host/PCI bridge's Subordinate Bus Number register to 1.
5. The enumeration software reads bridge A's Capability Register (Figure 21-5 on page 750 and Table 21 - 1 on page 753; a detailed description of this register can be found in "PCI Express Capabilities Register" on page 898). The value 0100b in the register's Device/Port Type field indicates that this a Root Port on the Root Complex.
6. The specification states that the enumeration software must perform a depth-first search, so before proceeding to discover additional functions/devices on bus 0, it must proceed to search bus 1.
7. Software reads the Vendor ID of bus 1, device 0, function 0. A valid Vendor ID is returned, indicating that bus 1, device 0, function 0 exists.
8. The Header Type field in the Header register contains the value one (0000001b) indicating that this is a PCI-to-PCI bridge. In addition, bit 7 is a 0, indicating that bridge C is a single-function device.

9. Bridge C's Capability Register contains the value 0101b in the Device/Port Type field indicating that this is the upstream Port on a switch.
10. Software now performs a series of configuration writes to set bridge C's bus number registers as follows:
    - Primary Bus Number Register = 1.
    - Secondary Bus Number Register = 2.
    - Subordinate Bus Number Register = 2.

    Bridge C is now aware that the number of the bus directly attached to its downstream side is 2 (Secondary Bus Number = 2) and the number of the bus farthest downstream of it is 2 (Subordinate Bus Number = 2).
11. Software updates the Subordinate Bus Number registers in the Host/PCI bridge and in bridge A to 2.
12. Continuing with its depth-first search, a read is performed from bus 2, device 0, function 0's Vendor ID register. The example assumes that bridge D is device 0, function 0 on bus 2.
13. A valid Vendor ID is returned, indicating that bus 2, device 0, function 0 exists.
14. The Header Type field in the Header register contains the value one (0000001b) indicating that this is a PCI-to-PCI bridge. In addition, bit 7 is a 0, indicating that bridge D is a single-function device.
15. Bridge D's Capability Register contains the value 0110b in the Device/Port Type field indicating that this is the downstream Port on a switch.
16. Software now performs a series of configuration writes to set bridge D's bus number registers as follows:
    - Primary Bus Number Register = 2.
    - Secondary Bus Number Register = 3.
    - Subordinate Bus Number Register = 3.

    Bridge D is now aware that the number of the bus directly attached to its downstream side is 3 (Secondary Bus Number = 3) and the number of the bus farthest downstream of it is 3 (Subordinate Bus Number = 3).
17. Software updates the Subordinate Bus Number registers in the Host/PCI bridge, bridge A, and bridge C to 3.
18. Continuing with its depth-first search, a read is performed from bus 3, device 0, function 0's Vendor ID register.
19. A valid Vendor ID is returned, indicating that bus 3, device 0, function 0 exists.
20. The Header Type field in the Header register contains the value zero (0000000b) indicating that this is an Endpoint device. In addition, bit 7 is a 1, indicating that this is a multifunction device.
21. The device's Capability Register contains the value 0000b in the Device/Port Type field indicating that this is an Endpoint device.
22. The enumeration software performs accesses to the Vendor ID of functions

1-through-7 in bus 3, device 0 and determines that only function 1 exists in addition to function 0.

23. Having exhausted the current leg of the depth first search, the enumeration software backs up one level (to bus 2) and moves on to read the Vendor ID of the next device (device 1). The example assumes that bridge E is device 1, function 0 on bus 2.
24. A valid Vendor ID is returned, indicating that bus 2, device 1, function 0 exists.
25. The Header Type field in bridge E's Header register contains the value one (0000001b) indicating that this is a PCI-to-PCI bridge. In addition, bit 7 is a 0, indicating that bridge E is a single-function device.
26. Bridge E's Capability Register contains the value 0110b in the Device/Port Type field indicating that this is the downstream Port on a switch.
27. Software now performs a series of configuration writes to set bridge E's bus number registers as follows:
    - Primary Bus Number Register = 2.
    - Secondary Bus Number Register = 4.
    - Subordinate Bus Number Register = 4.

    Bridge E is now aware that the number of the bus directly attached to its downstream side is 4 (Secondary Bus Number = 4) and the number of the bus farthest downstream of it is 4 (Subordinate Bus Number = 4).
28. Software updates the Subordinate Bus Number registers in the Host/PCI bridge, bridge A, and bridge C to 4.
29. Continuing with its depth-first search, a read is performed from bus 4, device 0, function 0's Vendor ID register.
30. A valid Vendor ID is returned, indicating that bus 4, device 0, function 0 exists.
31. The Header Type field in the Header register contains the value zero (0000000b) indicating that this is an Endpoint device. In addition, bit 7 is a 0, indicating that this is a single-function device.
32. The device's Capability Register contains the value 0000b in the Device/Port Type field indicating that this is an Endpoint device.
33. Having exhausted the current leg of the depth first search, the enumeration software backs up one level (to bus 2) and moves on to read the Vendor ID of the next device (device 2). The example assumes that devices 2-through-31 are not implemented on bus 2, so no additional devices are discovered on bus 2.
34. The enumeration software backs up to the bus within the Root Complex (bus 0) and moves on to read the Vendor ID of the next device (device 1). The example assumes that bridge B is device 1, function 0 on bus 0.
35. In the same manner as previously described, the enumeration software discovers bridge B and performs a series of configuration writes to set bridge

B's bus number registers as follows:

- Primary Bus Number Register = 0.
- Secondary Bus Number Register = 5.
- Subordinate Bus Number Register = 5.

Bridge B is now aware that the number of the bus directly attached to its downstream side is 5 (Secondary Bus Number = 5) and the number of the bus farthest downstream of it is 5 (Subordinate Bus Number = 5).

36. The Host/PCI's Subordinate Bus Number is updated to 5.
37. Bridge F is then discovered and a series of configuration writes are performed to set its bus number registers as follows:
    - Primary Bus Number Register = 5.
    - Secondary Bus Number Register = 6.
    - Subordinate Bus Number Register = 6.

    Bridge F is now aware that the number of the bus directly attached to its downstream side is 6 (Secondary Bus Number = 6) and the number of the bus farthest downstream of it is 6 (Subordinate Bus Number = 6).
38. The Host/PCI bridge's and bridge B' Subordinate Bus Number registers are updated to 6.
39. Bridge G is then discovered and a series of configuration writes are performed to set its bus number registers as follows:
    - Primary Bus Number Register = 6.
    - Secondary Bus Number Register = 7.
    - Subordinate Bus Number Register = 7.

    Bridge F is now aware that the number of the bus directly attached to its downstream side is 7 (Secondary Bus Number = 7) and the number of the bus farthest downstream of it is 7 (Subordinate Bus Number = 7).
40. The Host/PCI bridge's Subordinate Bus Number register is updated to 7. Bridge B's and F's Subordinate Bus Number registers are also updated to 7.
41. A single-function Endpoint device is discovered at bus 7, device 0, function 0.
42. Bridge H is then discovered and a series of configuration writes are performed to set its bus number registers as follows:
    - Primary Bus Number Register = 6.
    - Secondary Bus Number Register = 8.
    - Subordinate Bus Number Register = 8.

    Bridge F is now aware that the number of the bus directly attached to its downstream side is 8 (Secondary Bus Number = 8) and the number of the bus farthest downstream of it is 8 (Subordinate Bus Number = 8).
43. The Host/PCI bridge's Subordinate Bus Number register is updated to 8. Bridge B's and F's Subordinate Bus Number registers are also updated to 8.
44. Bridge J is discovered and its Capability register's Device/Port Type fields identifies it as a PCI Express-to-PCI bridge.
45. A series of configuration writes are performed to set bridge J's bus number registers as follows:

22 PCI Compatible Configuration Registers

The Previous Chapter

The previous chapter provides a detailed description of the discovery process and bus numbering. It described:

- Enumerating a system with a single Root Complex
- Enumerating a system with multiple Root Complexes
- A multifunction device within a Root Complex or a Switch
- An Endpoint embedded in a Switch or Root Complex
- Automatic Requester ID assignment.
- Root Complex Register Blocks (RCRBs)

This Chapter

This chapter provides a detailed description of the configuration registers residing a function's PCI-compatible configuration space. This includes the registers for both non-bridge and bridge functions.

The Next Chapter

The next chapter provides a detailed description of device ROMs associated with PCI, PCI Express, and PCI-X functions. This includes the following topics:

- device ROM detection.
- internal code/data format.
- shadowing.

- initialization code execution.
- interrupt hooking.

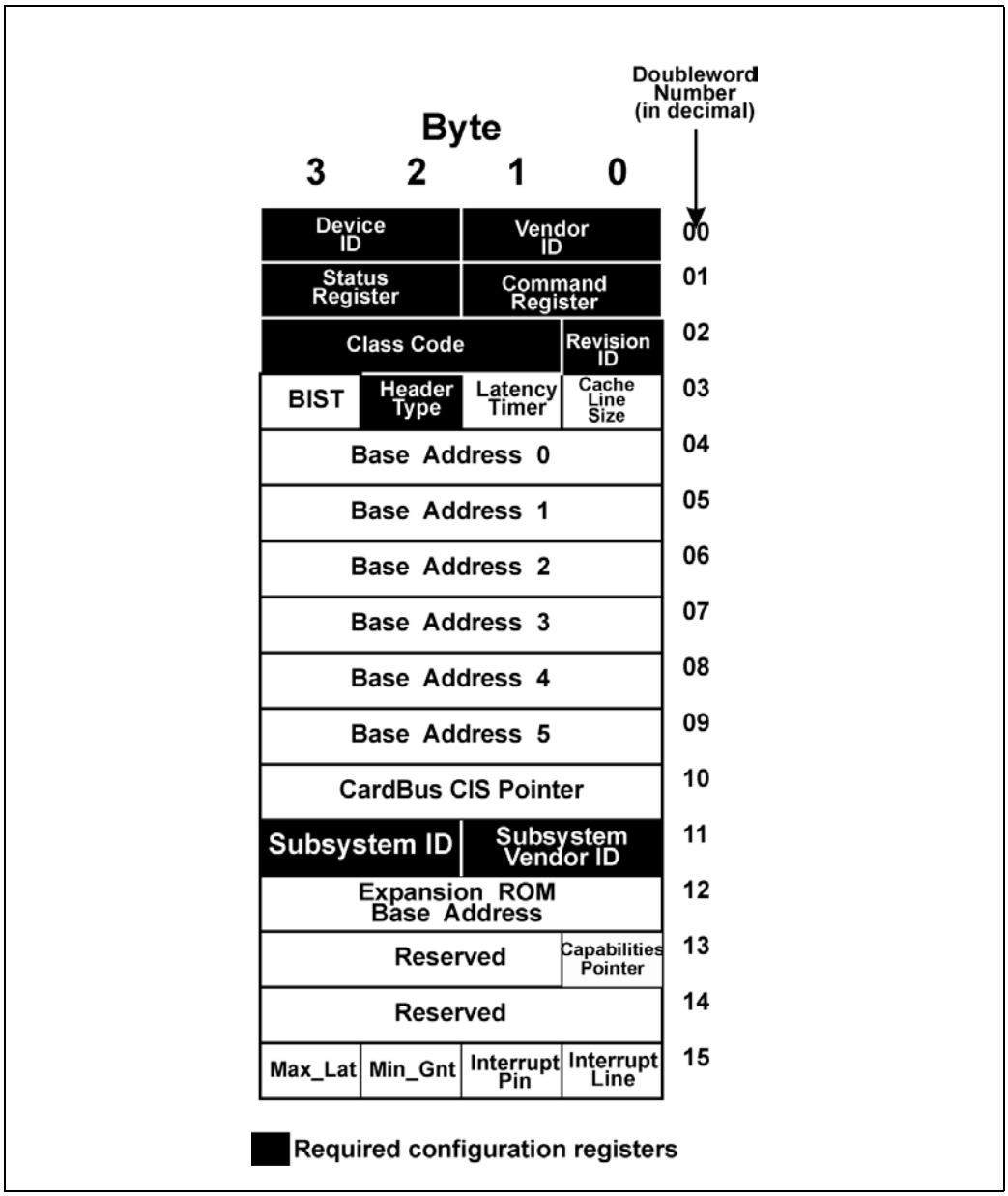
Header Type 0

General

Figure 22-1 on page 771 illustrates the format of a function's Header region (for functions other than PCI-to-PCI bridges and CardBus bridges). The registers marked in black are always mandatory. Note that although many of the configuration registers in the figure are not marked mandatory, a register may be mandatory for a particular type of device. The subsequent sections define each register and any circumstances wherein it may be mandatory.

As noted earlier, this format is defined as Header Type 0. The registers within the Header are used to identify the device, to control its functionality and to sense its status in a generic manner. The usage of the device's remaining 48 dwords of PCI-compatible configuration space is intended for device-specific registers, but, with the advent of the 2.2 PCI spec, is also used as an overflow area for some new registers defined in the PCI spec (for more information, refer to "Capabilities Pointer Register" on page 779).

Figure 22-1: Header Type 0



Header Type 0 Registers Compatible With PCI

The Header Type 0 PCI configuration registers that are implemented and used identically in PCI and PCI Express are:

- Vendor ID register.
- Device ID register.
- Revision ID register.
- Class Code register.
- Subsystem Vendor ID register.
- Subsystem ID register.
- Header Type register.
- BIST register.
- Capabilities Pointer register.
- CardBus CIS Pointer register.
- Expansion ROM Base Address register.

The sections that follow provide a description of each of these registers.

Header Type 0 Registers Incompatible With PCI

In a non-bridge PCI Express function, the definitions of the following configuration registers in the function's PCI-compatible configuration space differ from the PCI spec's definition of the respective register definitions:

- Command Register
- Status Register
- Cache Line Size Register
- Master Latency Timer Register
- Interrupt Line Register
- Interrupt Pin Register
- Base Address Registers
- Min_Gnt/Max_Lat Registers

The sections that follow define the implementation/usage differences of these registers. For a full description of their implementation in a PCI function, refer to the MindShare book entitled *PCI System Architecture, Fourth Edition* (published by Addison-Wesley). For a full description of their implementation in a PCI-X function, refer to the MindShare book entitled *PCI-X System Architecture*, First Edition (published by Addison-Wesley).

Registers Used to Identify Device's Driver

The OS uses some combination of the following mandatory registers to determine which driver to load for a device:

- Vendor ID.
- Device ID.
- Revision ID.
- Class Code.
- SubSystem Vendor ID.
- SubSystem ID.

Vendor ID Register

PCI-Compatible register. Always mandatory. This 16-bit register identifies the manufacturer of the function. The value hardwired in this read-only register is assigned by a central authority (the PCI SIG) that controls issuance of the numbers. The value FFFFh is reserved and must be returned by the Host/PCI bridge when an attempt is made to perform a configuration read from a configuration register in a non-existent function. In PCI or PCI-X, the read attempt results in a Master Abort, while in PCI Express it results in the return of UR (Unsupported Request) completion status. In either case, the bridge must return a Vendor ID of FFFFh. The error status returned is not considered to be an error, but the specification says that the bridge must nonetheless set its Received Master Abort bit in its configuration Status register.

Device ID Register

PCI-Compatible register. Always mandatory. This 16-bit value is assigned by the function manufacturer and identifies the type of function. In conjunction with the Vendor ID and possibly the Revision ID, the Device ID can be used to locate a function-specific (and perhaps revision-specific) driver for the function.

Revision ID Register

PCI-Compatible register. Always mandatory. This 8-bit value is assigned by the function manufacturer and identifies the revision number of the function. If the vendor has supplied a revision-specific driver, this is handy in ensuring that the correct driver is loaded by the OS.

Class Code Register

General. *PCI-Compatible register. Always mandatory.* The Class Code register is pictured in Figure 22-2 on page 775. It is a 24-bit, read-only register divided into three fields: base Class, Sub Class, and Programming Interface. It identifies the basic function of the function (e.g., a mass storage controller), a more specific function sub-class (e.g., IDE mass storage controller), and, in some cases, a register-specific programming interface (such as a specific flavor of the IDE register set).

- The upper byte defines the base Class of the function,
- the middle byte defines a sub-class within the base Class,
- and the lower byte defines the Programming Interface.

The currently-defined base Class codes are listed in Table 22-1 on page 775. Table 2 on page 1020 through Table 19 on page 1031 define the Subclasses within each base Class. For many Class/SubClass categories, the Programming Interface byte is hardwired to return zeros (in other words, it has no meaning). For some, such as VGA-compatible functions and IDE controllers, it does have meaning.

This register is useful when the OS is attempting to locate a function that a Class driver can work with. As an example, assume that a particular device driver has been written to work with any display adapter that is 100% XGA register set-compatible. If the OS can locate a function with a Class of 03h (see Table 22-1 on page 775) and a Sub Class of 01h (see Table 5 on page 1022), the driver will work with that function. A Class driver is more flexible than a driver that has been written to work only with a specific function from a specific vendor.

The Programming Interface Byte. For some functions (such as the XGA display adapter used as an example in the previous section) the combination of the Class Code and Sub Class Code is sufficient to fully-define its level of register set compatibility. The register set layout for some function types, however, can vary from one implementation to another. As an example, from a programming interface perspective there are a number of flavors of IDE mass storage controllers, so it's not sufficient to identify yourself as an IDE mass storage controller. The Programming Interface byte value (see Table 20 on page 1031) provides the final level of granularity that identifies the exact register set layout of the function.

23 Expansion ROMs

The Previous Chapter

The previous chapter provided a detailed description of the configuration registers residing a function's PCI-compatible configuration space. This included the registers for both non-bridge and bridge functions.

This Chapter

This chapter provides a detailed description of device ROMs associated with PCI, PCI Express, and PCI-X functions. This includes the following topics:

- device ROM detection.
- internal code/data format.
- shadowing.
- initialization code execution.
- interrupt hooking.

The Next Chapter

The next chapter provides a description of:

- The PCI Express Capability register set in a function's PCI-compatible configuration space.
- The optional PCI Express Extended Capabilities register sets in a function's extended configuration space:
  - The Advanced Error Reporting Capability register set.
  - Virtual Channel Capability register set.
  - Device Serial Number Capability register set.
  - Power Budgeting Capability register set.
- RCRBs.

ROM Purpose—Device Can Be Used In Boot Process

In order to boot the OS into memory, the system needs three devices:

- A mass storage device to load the OS from. This is sometimes referred to as the **IPL** (Initial Program Load) **device** and is typically an IDE or a SCSI hard drive.
- A display adapter to enable progress messages to be displayed during the boot process. In this context, this is typically referred to as the **output device**.
- A keyboard to allow the user to interact with the machine during the boot process. In this context, this is typically referred to as the **input device**.

The OS must locate three devices that fall into these categories and **must also locate a device driver associated with each of the devices**. Remember that the OS hasn't been booted into memory yet and therefore hasn't loaded any loadable device drivers into memory from disk! This is the main reason that device ROMs exist. It contains a device driver that permits the device to be used during the boot process.

ROM Detection

When the configuration software is configuring a PCI, PCI-X, or PCI-Express function, it determines if a function-specific ROM exists by checking to see if the designer has implemented an Expansion ROM Base Address Register (refer to Figure 23-1 on page 873).

As described in "Expansion ROM Base Address Register" on page 783, the programmer writes all ones (with the exception of bit zero, to prevent the enabling of the ROM address decoder; see Figure 23-1 on page 873) to the Expansion ROM Base Address Register and then reads it back. If a value of zero is returned, then the register is not implemented and there isn't an expansion ROM associated with the device.

On the other hand, the ability to set any bits to ones indicates the presence of the Expansion ROM Base Address Register. This may or may not indicate the presence of a device ROM. Although the address decoder and a socket may exist for a device ROM, the socket may not be occupied at present. The programmer determines the presence of the device ROM by:

- assigning a base address to the register's Base Address field,
- enabling its decoder (by setting bit 0 in the register to one),
- and then attempting to read the first two locations from the ROM.

If the first two locations contain the ROM signature—AA55h—then the ROM is present.

Figure 23-1 on page 873 illustrates the format of the Expansion ROM Base Address Register. Assume that the register returns a value of FFFE0000h when read back after writing all ones to it. Bit 17 is the least-significant bit that was successfully changed to a one and has a binary-weighted value of 128K. This indicates that it is a 128KB ROM decoder and bits [24:17] within the Base Address field are writable. The programmer now writes a 32-bit start address into the register and sets bit zero to one to enable its ROM address decoder. The function's ROM address decoder is then enabled and the ROM (if present) can be accessed. The maximum ROM decoder size permitted by the PCI spec is 16MB, dictating that bits [31:25] must be read/write.

The programmer then performs a read from the first two locations of the ROM and checks for a return value of AA55h. If this pattern is not received, the ROM is not present. The programmer disables the ROM address decoder (by clearing bit zero of the Expansion ROM Base Address Register to zero). If AA55h is received, the ROM exists and a device driver code image must be copied into main memory and its initialization code must be executed. This topic is covered in the sections that follow.

Figure 23-1: Expansion ROM Base Address Register Bit Assignment

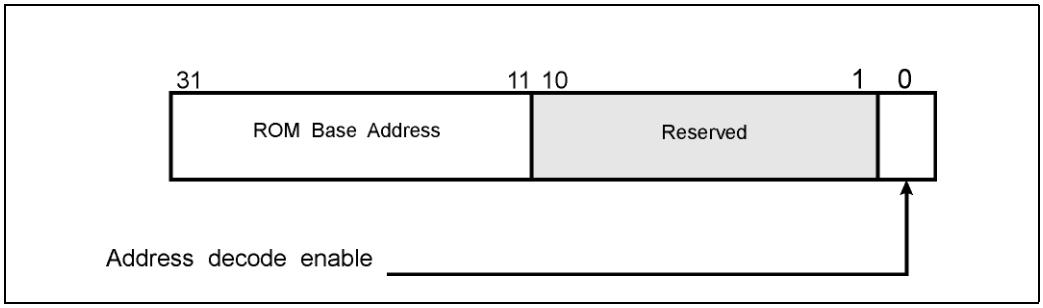
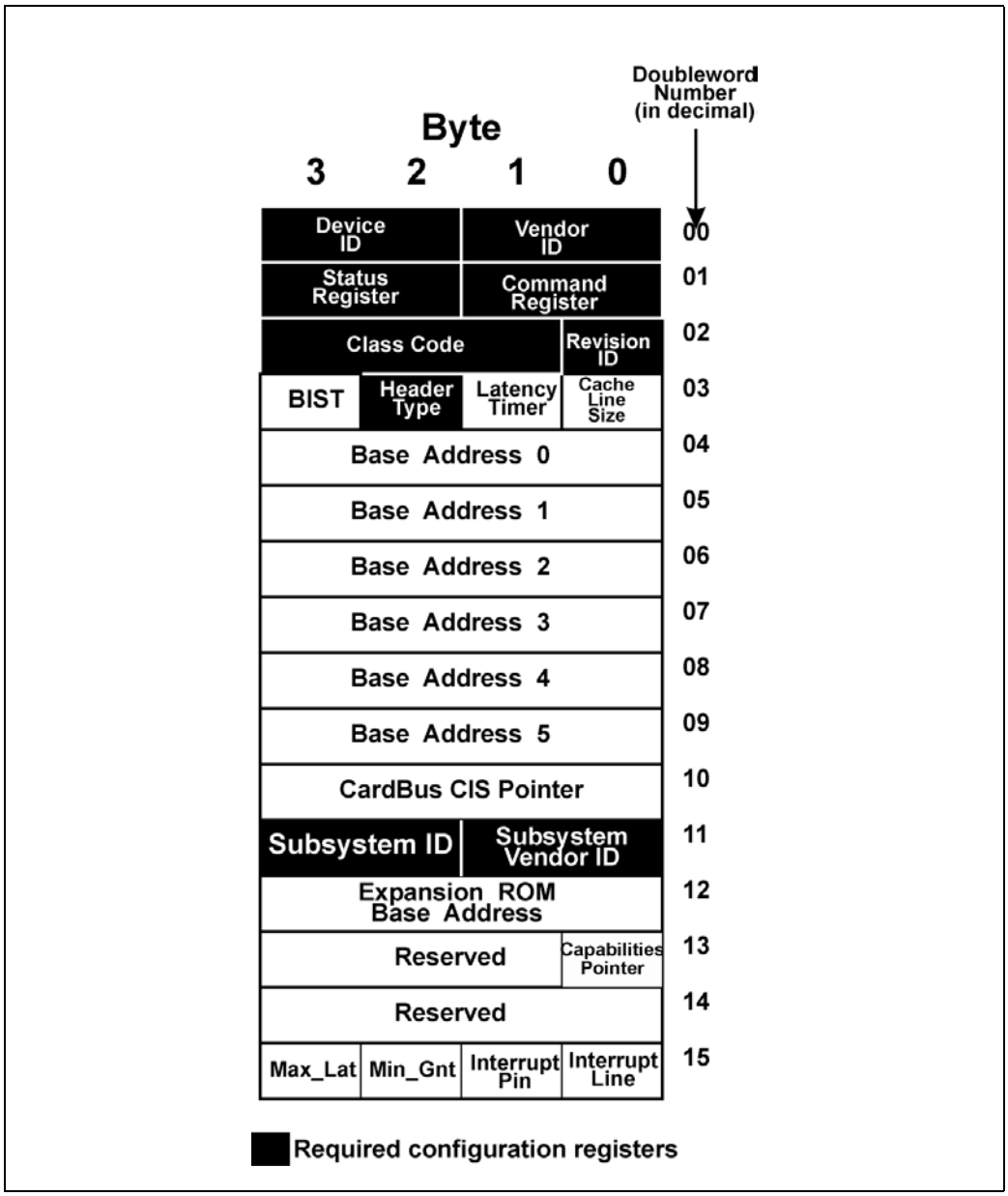


Figure 23-2: Header Type Zero Configuration Register Format



ROM Shadowing Required

The PCI spec requires that device ROM code is never executed in place (i.e., from the ROM). It must be copied to main memory. This is referred to as "shadowing" the ROM code. This requirement exists for two reasons:

- ROM access time is typically quite slow, resulting in poor performance whenever the ROM code is fetched for execution.
- Once the initialization portion of the device driver in the ROM has been executed, it can be discarded and the code image in main memory can be shortened to include only the code necessary for run-time operation. The portion of main memory allocated to hold the initialization portion of the code can be freed up, allowing more efficient use of main memory.

Once the presence of the device ROM has been established (see the previous section), the configuration software must copy a code image into main memory and then disable the ROM address decoder (by clearing bit zero of the Expansion ROM Base Address Register to zero). In a non-PC environment, the area of memory the code image is copied to could be anywhere in memory space. The specification for that environment may define a particular area.

In a PC environment, the ROM code image must be copied into main memory into the range of addresses historically associated with device ROMs: 000C0000h through 000DFFFFh. If the Class Code indicates that this is the VGA's device ROM, its code image must be copied into memory starting at location 000C0000h.

The next section defines the format of the information in the ROM and how the configuration software determines which code image (yes, there can be more than one device driver) to load into main memory.

ROM Content

Multiple Code Images

The PCI spec permits the inclusion of more than one code image in a PCI device ROM. Each code image would contain a copy of the device driver in a specific machine code, or in interpretive code (explained later). The configuration software can then scan through the images in the ROM and select the one best

suited to the system processor type. The ROM might contain drivers for various types of devices made by this device's vendor. The code image copied into main memory should match up with the function's ID. To this end, each code image also contains:

- the Vendor ID and Device ID. This is useful for matching up the driver with a function that has a vendor/device match.
- the Class Code. This is useful if the driver is a Class driver that can work with any compatible device within a Class/SubClass. For more information, see "Class Code Register" on page 774.

Figure 23-3 on page 877 illustrates the concept of multiple code images embedded within a device ROM. Each image must start on an address evenly-divisible by 512. Each image consists of two data structures, as well as a run-time code image and an initialization code image. The configuration software interrogates the data structures in order to determine if this is the image it will copy to main memory and use. If it is, the configuration software:

1. Copies the image to main memory,
2. Disables the expansion ROM's address decoder,
3. Executes the initialization code,
4. If the initialization code shortens the length indicator in the data structure, the configuration software deallocates the area of main memory that held the initialization portion of the driver (in Figure 23-4 on page 879, notice that the initialization portion of the driver is always at the end of the image).
5. The area of main memory containing the image is then write-protected.

The sections that follow provide a detailed discussion of the code image format and the initialization process.

24 Express-Specific Configuration Registers

The Previous Chapter

The previous chapter provided a detailed description of device ROMs associated with PCI, PCI Express, and PCI-X functions. This included the following topics:

- device ROM detection.
- internal code/data format.
- shadowing.
- initialization code execution.
- interrupt hooking.

This Chapter

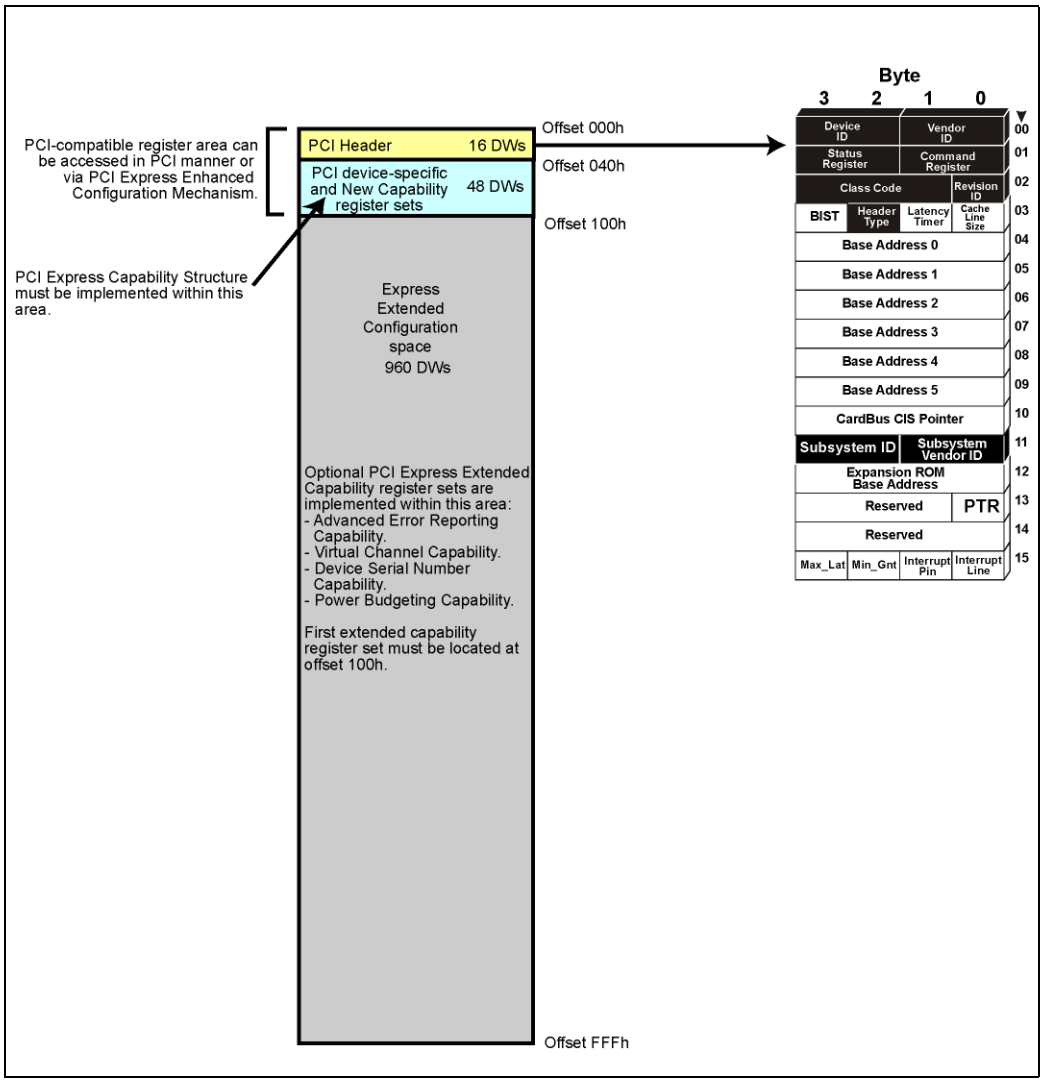
This chapter provides a description of:

- The PCI Express Capability register set in a function's PCI-compatible configuration space.
- The optional PCI Express Extended Capabilities register sets in a function's extended configuration space:
  - The Advanced Error Reporting Capability register set.
  - Virtual Channel Capability register set.
  - Device Serial Number Capability register set.
  - Power Budgeting Capability register set.
- RCRBs.

Introduction

Refer to Figure 24-1 on page 895. As described earlier in "Each Function Implements a Set of Configuration Registers" on page 715, each PCI Express function has a dedicated 4KB memory address range within which its configuration registers are implemented. Each Express function must implement the PCI Express Capability register set somewhere in the lower 48 dwords of the PCI-compatible register space (i.e., within the lower 48 dword region of the first 64 dwords of configuration space). In addition, the function may optionally implement any of the PCI Express Extended Capability register sets. The sections that follow provide a detailed description of each of these Express-specific register sets.

Figure 24-1: Function's Configuration Space Layout



PCI Express Capability Register Set

Introduction

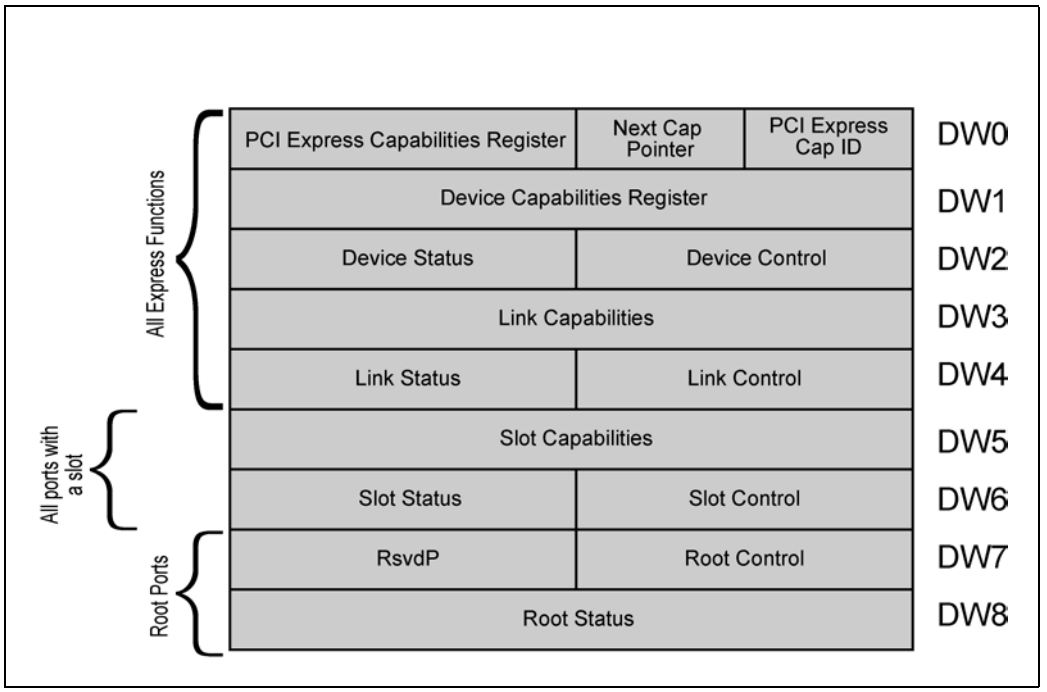
Refer to Figure 24-2 on page 897. Otherwise referred to as the PCI Express Capability Structure, implementation of the PCI Express Capability register set is mandatory for each function. It is implemented as part of the linked list of Capability register sets that reside in the lower 48 dwords of a function's PCI-compatible register area. It should be noted however, that some portions of this register set are optional.

Register implementation requirements:

- Every Express function must implement the registers that reside in dwords 0-through-4.
- The bridge associated with each Root Port must implement the registers that reside in dwords seven and eight.
- Each bridge associated with a Root Port or a downstream Switch Port that is connected to a slot (i.e., an add-in card slot) must implement the registers that reside in dwords five and six.

The sections that follow provide a detailed description of each of these registers.

Figure 24-2: PCI Express Capability Register Set



Required Registers

General

The sections that follow describe each of the required registers within the PCI Express Capability register set. The following registers must be implemented by all Express functions:

- PCI Express Capability ID Register
- Next Capability Pointer Register
- PCI Express Capabilities Register
- Device Capabilities Register
- Device Control Register
- Device Status Register
- Link Capabilities Register
- Link Control Register
- Link Status Register

PCI Express Capability ID Register

This read-only field must contain the value 10h, indicating this is the start of the PCI Express Capability register set.

Next Capability Pointer Register

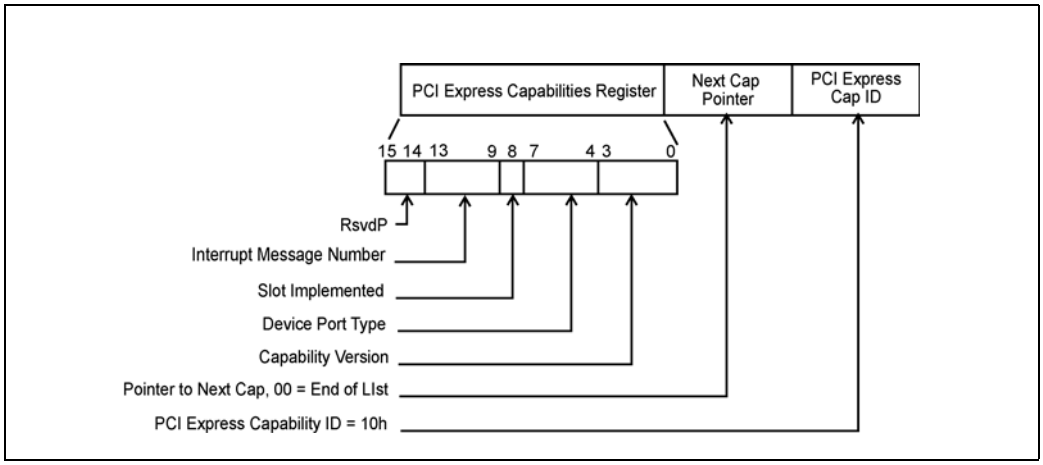
This read-only field contains one of the following:

- The dword-aligned, non-zero offset to the next capability register set in the lower 48 dwords of the function's PCI-compatible configuration space.
- 00h, if the PCI Express Capability register set is the final register set in the linked list of capability register sets in the function's PCI-compatible configuration space.

PCI Express Capabilities Register

Figure 24-3 on page 898 illustrates this register and Table 24 - 1 on page 899 provides a description of each bit field in this register.

Figure 24-3: PCI Express Capabilities Register



Appendix A
Test, Debug and Verification of PCI Express™ Designs

by Gordon Getty, Agilent Technologies

Scope

The need for greater I/O bandwidth in the computer industry has caused designers to shift from using parallel buses like ISA, PCI™ and PCI-X™ to using multi-lane serial interconnects running at Gigabit speed. The industry has settled on PCI Express™ technology as the key I/O technology of the future, as it delivers on the higher bandwidth requirements, helps to reduce cost for silicon vendors and leverages the software environment from the pervasive PCI/PCI-X technology. While the change from parallel buses to multi-lane serial buses sounds like a small step, it presented a whole set of new debug and validation challenges to designers.

Serial technology requires a different approach to testing, starting from the physical layer and moving up through the transaction layer. In many cases, the parallel bus had several slots connected to the same physical lines, which allowed you to connect test equipment to the same bus and monitor other devices. With the point-to-point nature of serial technologies, this is no longer possible, and with the speed moving from the megahertz range to the gigahertz range, probing of the signal becomes a real challenge.

The second generation of PCI Express, known as PCI Express 2.0 (PCIe™ 2.0), is based on PCI Express 1.0 principles, but it supports speeds of up to 5 GT/s. Preserving backwards compatibility with PCI Express 1.0 presents its own set of challenges. Also, new and extended capabilities related to energy savings - including active state power management (ASPM) and dynamic link width negotiation - makes achieving interoperability between devices more challenging, especially if these features are implemented incorrectly. Careful design and validation processes can help you avoid costly chip re-spins to fix interoperability issues.

This chapter will guide you through overcoming the challenges faced when you debug and validate your PCI Express devices.

Electrical Testing at the Physical Layer

PCI Express specification requires devices to have a built-in mechanism for testing the electrical characteristics of the devices, such as exists on motherboards and systems. When the transmit lanes of a device are terminated with a 50-ohm load, the transmit lanes are forced into a special mode known as compliance mode.

When a device is in compliance mode, it automatically generates a specific pattern known as the compliance pattern. Two different de-emphasis modes are introduced with the 5.0 Gb/s transfer rate. All add-in cards should be tested at the 2.5 Gb/s speed with -3.5 dB de-emphasis (Figure A-1), and at 5.0 Gb/s (Figure A-2) with both -3.5 dB de-emphasis and -6 dB de-emphasis.

Figure A-1: 2.5-GT/s PCIe Compliance Pattern

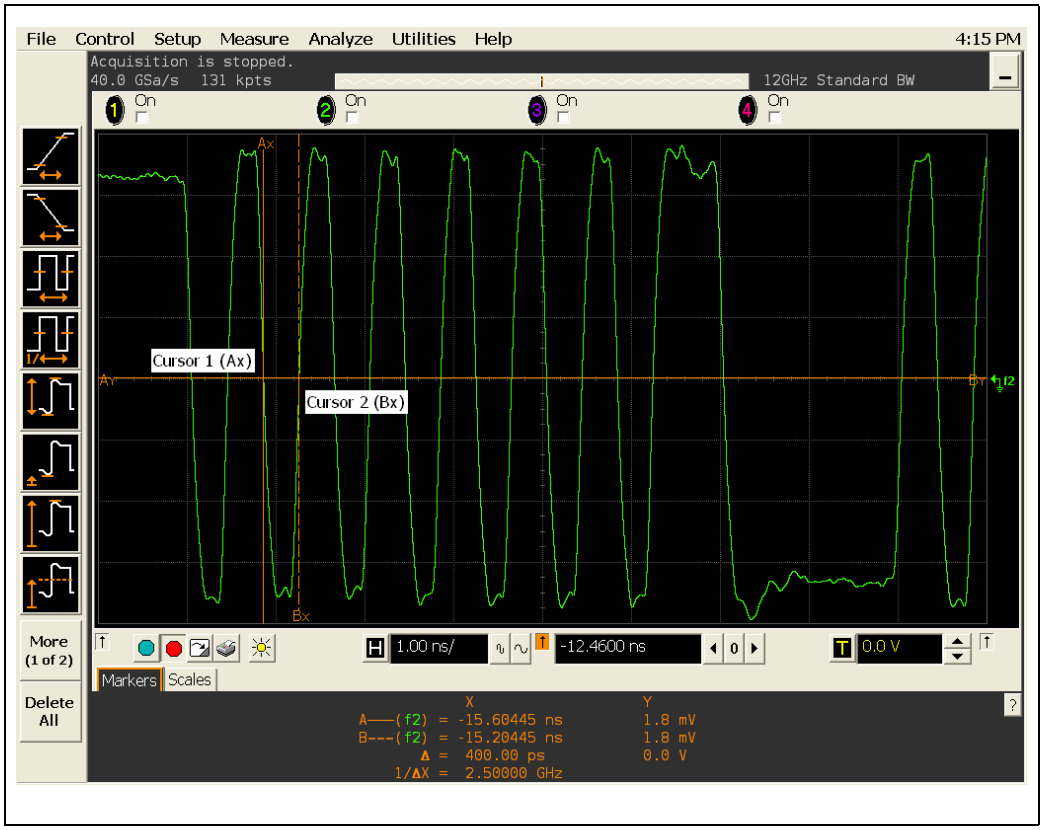
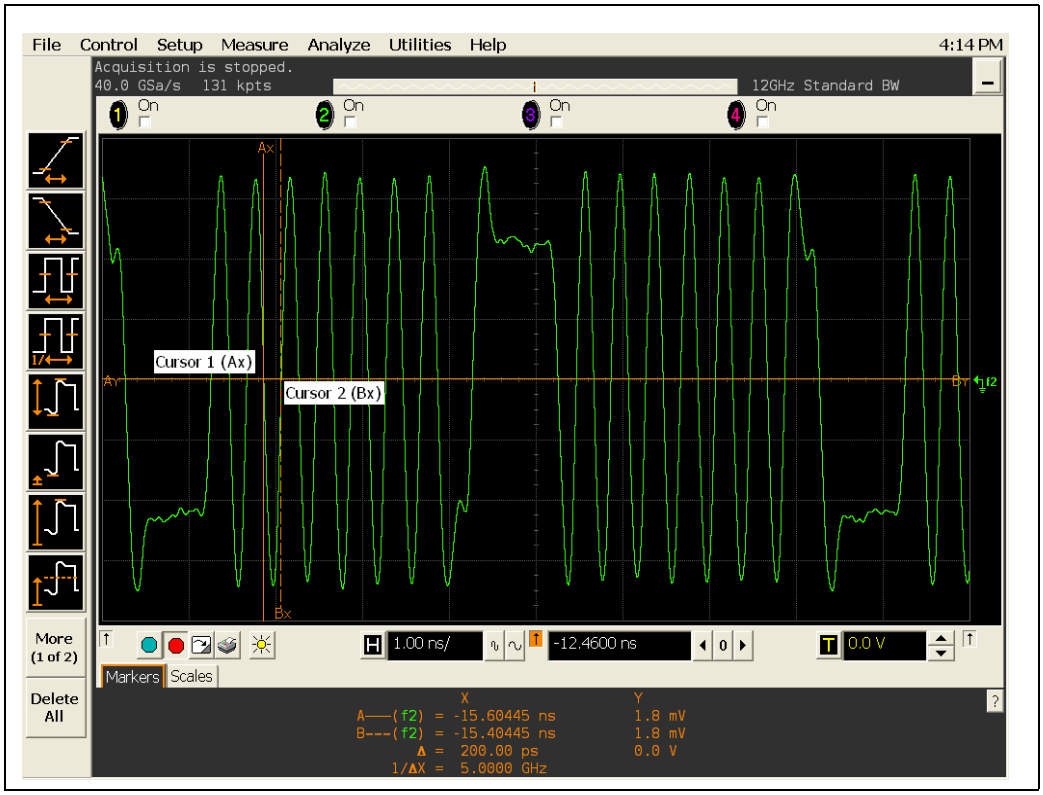


Figure A-2: 5-GT/s PCIe Compliance Pattern



The equipment required to carry out electrical testing on PCIe 2.0 devices includes a high-performance oscilloscope such as the Agilent Technologies DSO81304B 13-GHz Infiniium scope and a board into which you can plug an add-in card to provide a load on its transmitters. Alternatively, you can use a load board that plugs into a system and forces its transmitters into compliance mode ensuring that the device is generating a measurable signal.

PCI Express specifications (V1.1 and later) requires you to capture and process one million unit intervals of data to be able to make a valid measurement. The Agilent 81304B scope has a “QuickMeas” (QM) function that provides user-defined macros and data capture functionality intended to meet needs that may be very specific to a given application or measurement.

The PCI-SIG® provides compliance base board and compliance load board to help accomplish these tasks. These boards provide a consistent platform to make electrical measurements. Figures A-3 and A-4 show a typical setup.

Figure A-3: Typical Setup for Testing and Add-In Card

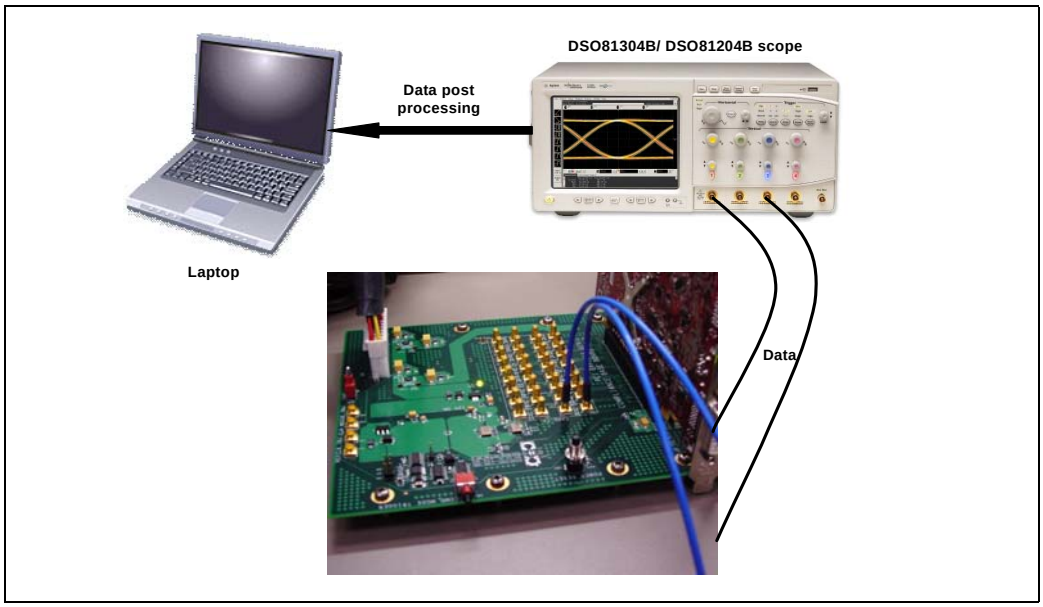
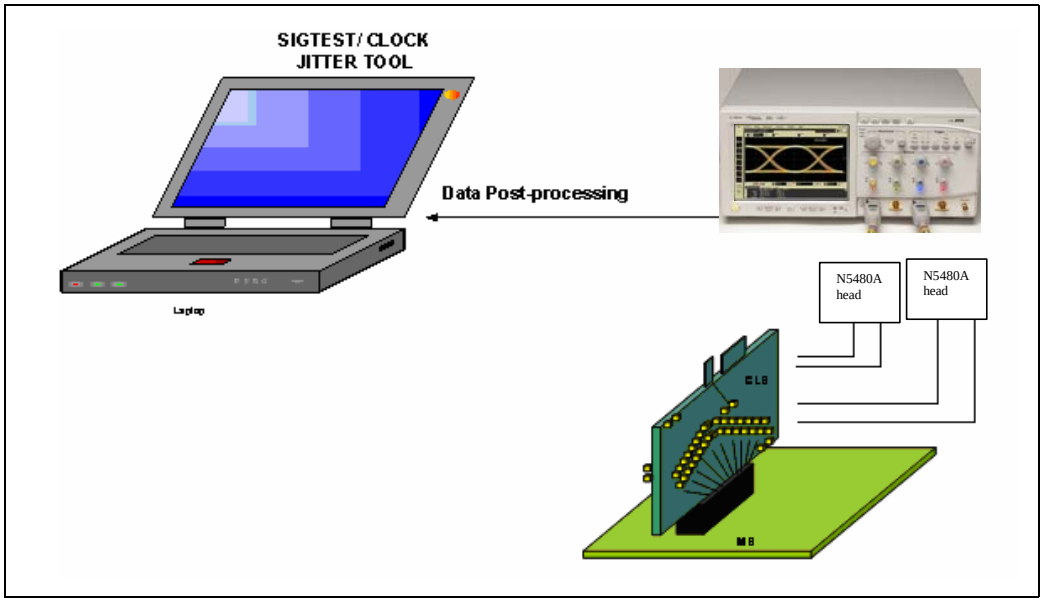
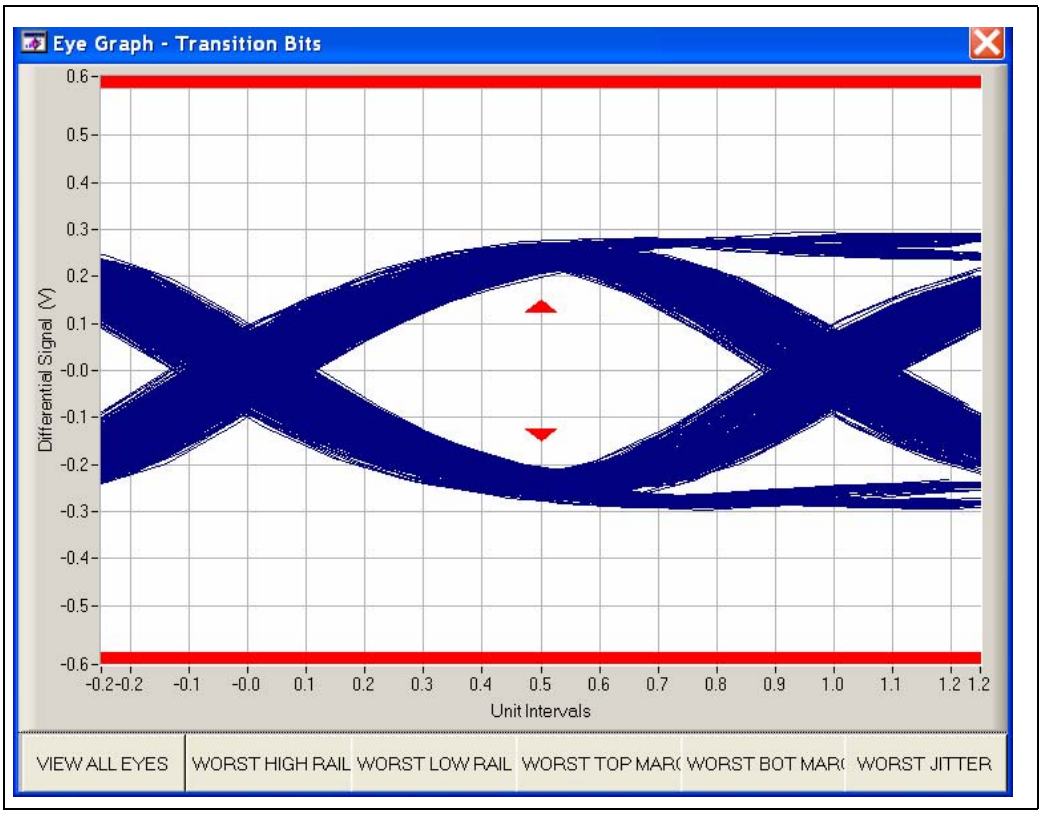


Figure A-4: Typical Setup for Testing a Motherboard



With the setups shown in Figures A-3 and A-4, data is captured on the oscilloscope. Post-processing is used to measure jitter on the reference clock and to measure the random and deterministic jitter on the data lines. In electrical testing, you need to test each individual lane independently, as each lane is likely to have different electrical characteristics. The data is captured and then post-processed to form an eye diagram, such as the one shown in Figure A-5.

Figure A-5: Oscilloscope Eye Diagram



Using the eye diagram, you can measure the tolerances of voltage and jitter against the specification to determine if the device is compliant electrically. If you find the device is not compliant, you have an early indicator that interoperability is a potential issue.

Appendix B

Markets & Applications for the PCI Express™ Architecture

By Larry Chisvin, Akber Kazmi, and Danny Chi (PLX Technology, Inc.)

Introduction

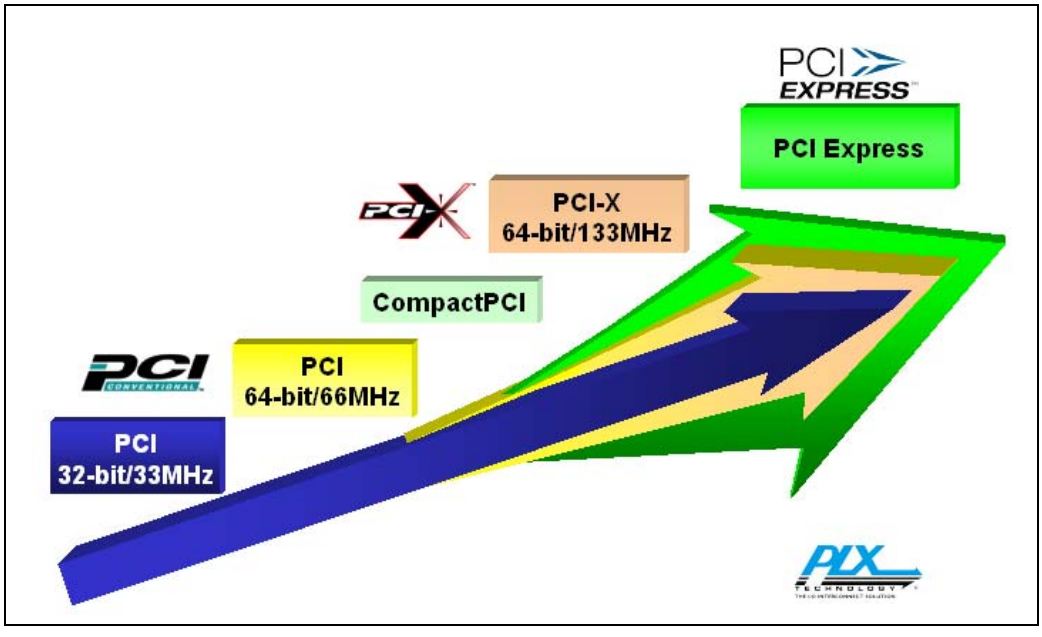
Since its definition in the early 1990's, PCI has become one of the most successful interconnect technologies ever used in computers. Originally intended for personal computer systems, the PCI architecture has penetrated into virtually every computing platform category, including servers, storage, communications, and a wide range of embedded control applications. From its early incarnation as a 32-bit 33MHz interconnect, it has been expanded to offer higher speeds (currently in widespread use at 64-bit 133MHz, with faster versions on the way). Most importantly, each advancement in PCI bus speed and width provided backward software compatibility, allowing designers to leverage the broad code base.

As successful as the PCI architecture has become, there is a limit to what can be accomplished with a multi-drop, parallel shared bus interconnect technology. Issues such as clock skew, high pin count, trace routing restrictions in printed circuit boards (PCB), bandwidth and latency requirements, physical scalability, and the need to support Quality of Service (QoS) within a system for a wide variety of applications lead to the definition of the PCI Express™ architecture.

PCI Express is the natural successor to PCI, and was developed to provide the advantages of a state-of-the-art, high-speed serial interconnect technology and packet based layered architecture, but maintain backward compatibility with the large PCI software infrastructure. The key goal was to provide an opti-

mized and universal interconnect solution for a great variety of future platforms, including desktop, server, workstation, storage, communications and embedded systems.

Figure B-1: Migration from PCI to PCI Express



This chapter provides an overview of the markets and applications that PCI Express is expected to serve, with an explanation of how the technology will be integrated into each application, and some exploration of the advantages that PCI Express brings to each usage.

Let's review the key benefits of the PCI Express architecture before we discuss its application in different markets. Some of the key features of the architecture we reviewed in this book are:

- Packet-based layered architecture
- Serial interconnection at 2.5 GHz (5 GHz being considered)
- Link-to-link and end-to-end error detection (CRC check)
- Point-to-point data flow
- Differential low voltage signals for noise immunity

- Quality of Service (QoS)and Virtual Channels (VC)
- Scalable from 1x to 32x lanes
- Software (backward) compatibility with legacy PCI systems

Enterprise Computing Systems

PCI Express is expected to be deployed initially in desktop and server systems. These computers typically utilize a chipset solution that includes one or more microprocessors and two types of special interconnect devices, called northbridges and southbridges. Northbridges connect the CPU with memory, graphics and I/O. Southbridges connect to standardized I/O devices such as hard disk drives, networking modules or devices, and often PCI expansion slots.

Desktop Systems

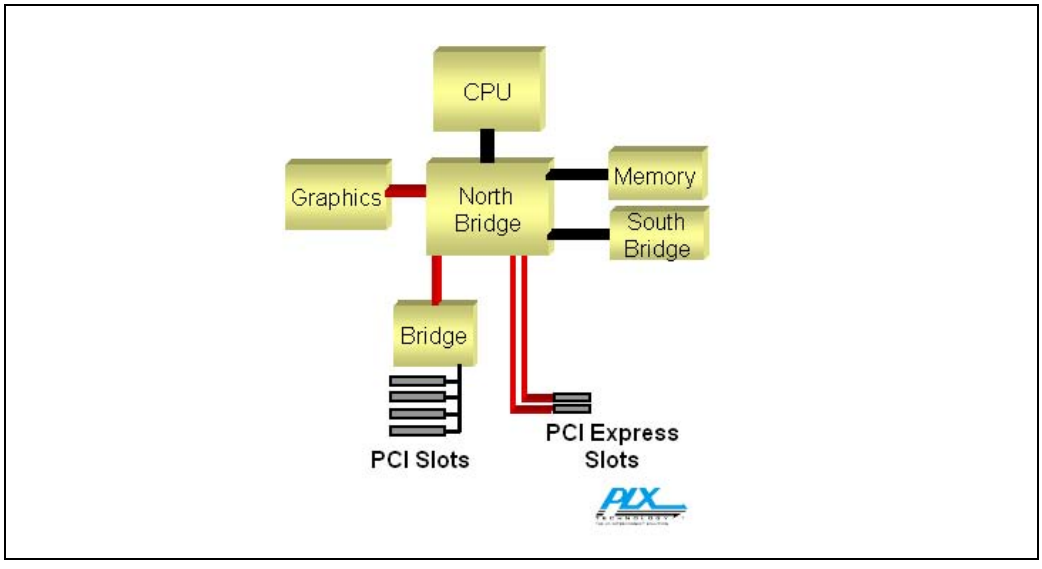
Typical use of PCI Express in a desktop application is shown in Figure B-2 on page 992. The PCI Express ports come directly out of the northbridge, and are bridged to PCI slots that are used for legacy plug-in cards. In some implementations the PCI Express interconnections will be completely hidden from the user behind PCI bridges, and in other implementations there will be PCI Express slots in a new PCI Express connector form factor.

The major benefit for using PCI Express in this application is the low pin count associated with serial interface technology, which will translate into lower cost. This low pin count provides the ability to create northbridges and I/O bridges with smaller footprints, and a significantly fewer number of board traces between the components. This provides a major reduction in the area and complexity of the signal/trace routing in PCBs.

Server Systems

Figure B-3 on page 993 shows PCI Express used in an enterprise server system. This system has similarities to the desktop system, since there is a northbridge and southbridge providing functions that parallel their roles in the desktop system, and the form factor of the system is often similar. Servers, however, place a greater emphasis on performance than desktop systems do.

Figure B-2: PCI Express in a Desktop System



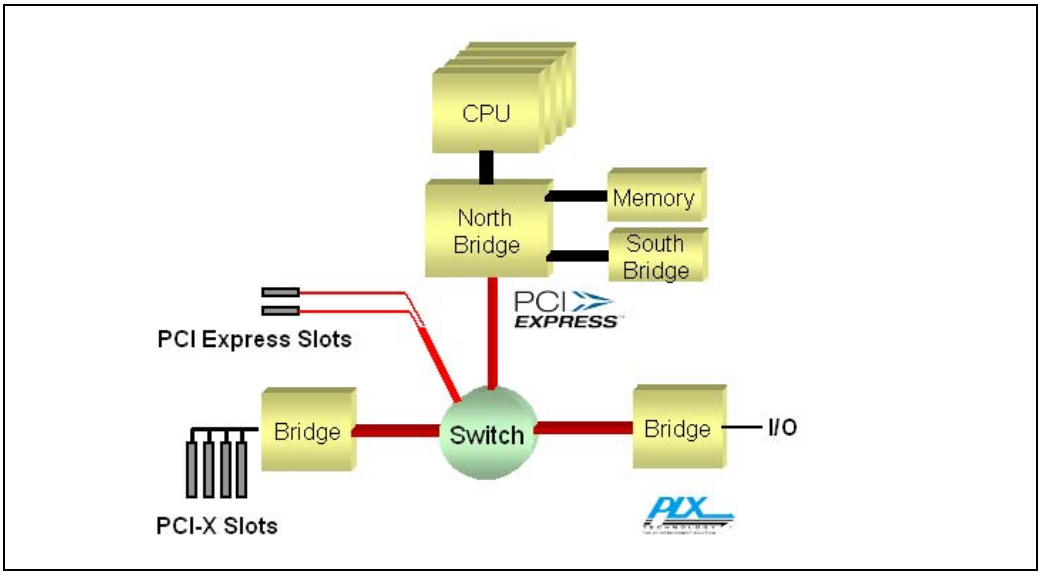
To achieve their performance and time to market objectives, server designers have adopted PCI-X. The primary attraction to PCI-X has been increased throughput, but with PCI code compatibility. PCI-X offers clear benefits compared to PCI, and will remain in server systems for a long while, but it suffers from the same shared bus limitations that have already been discussed. The high throughput of PCI Express serial interconnection provides a measurable benefit versus legacy interconnect technologies, especially as the speed of the I/O interconnect and the number of high speed I/O ports on each card increases.

Some systems will only provide PCI-X slots, but many newer systems will also offer several PCI Express slots. The number of PCI Express slots will grow over time compared to the PCI-X slots, and eventually will become dominant in the same way that PCI did with previous interconnect technologies. Since bandwidth is a primary motivator for a server, typical PCI Express slots will be either x4 or x8 lanes.

In most low to midrange server systems, the PCI-X bridging and PCI Express slots will be provided by using the ports right off of the northbridge. However, high-end systems will require more I/O slots of both kinds. Since PCI Express is a point-to-point technology, the only way to provide additional connection links is through a device called a fan out switch. Specifically, the purpose of a

fan out switch is to multiply the number of PCI Express lanes from an upstream host port to a higher number of downstream PCI Express devices. Figure 3 below, shows a PCI Express switch used in the system for this purpose.

Figure B-3: PCI Express in a Server System



Embedded Control

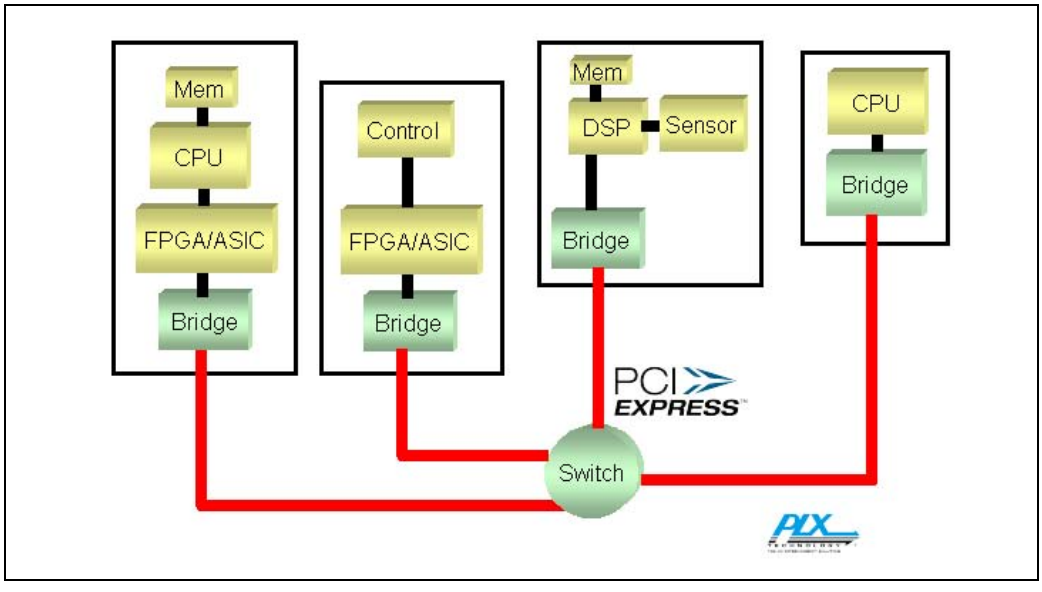
One of the many areas that PCI has penetrated is embedded-control systems. This describes a wide range of applications that measure, test, monitor, or display data, and includes applications such as industrial control, office automation, test equipment, and imaging.

In these applications, system designers typically utilize embedded processors. In many instances, leading-edge companies will differentiate their products by utilizing some custom logic in the form of an ASIC or FPGA. A bridge is often used to translate the simple custom interface and connect it to the bus.

It is expected that the embedded-control market will quickly migrate to PCI Express, with a typical example shown in Figure B-4 on page 994. Applications such as imaging and video streaming are always hungry for bandwidth, and the additional throughput of x4 or x8 PCI Express links will translate into

higher video resolution, or the handling of more video streams by the system. Others will implement PCI Express because of the noise resistance its LVDS traces provide, or because of its efficient routing and its ability to hook together subsystems through a standard cable. Still others will choose PCI Express simply because of its ubiquity.

Figure B-4: PCI Express in Embedded-Control Applications



Storage Systems

PCI has become a common backplane technology for mainstream storage systems. Although it provides a good mix of features, low cost, and throughput, the "bus" has become a performance bottleneck. Figure B-5 on page 995 shows the use of PCI Express in a storage system. Systems similar to the one shown in Figure B-5 on page 995 can be built on a motherboard, or as part of a backplane. The discussion in this section applies to both form factors.

We have highlighted increased bandwidth as one of the advantages of moving to PCI Express, and nowhere is it more beneficial and obvious than in storage. The bandwidth demanded by I/O connections such as Ethernet, Fibre Channel, SCSI, and InfiniBand, is increasing rapidly. And the ability to move data between I/O modules and the host processor is critical to overall system performance.

Appendix C

Implementing Intelligent Adapters and Multi-Host Systems With PCI Express™ Technology

By Jack Regula, Danny Chi and Tim Canepa (PLX Technology, Inc.)

Introduction

Intelligent adapters, host failover mechanisms and multiprocessor systems are three usage models that are common today, and expected to become more prevalent as market requirements for next generation systems. Despite the fact that each of these was developed in response to completely different market demands, all share the common requirement that systems that utilize them require multiple processors to co-exist within the system. This appendix outlines how PCI Express can address these needs through non-transparent bridging.

Because of the widespread popularity of systems using intelligent adapters, host failover and multihost technologies, PCI Express silicon vendors must provide a means to support them. This is actually a relatively low risk endeavor; given that PCI Express is software compatible with PCI, and PCI systems have long implemented distributed processing. The most obvious approach, and the one that PLX espouses, is to emulate the most popular implementation used in the PCI space for PCI Express. This strategy allows system designers to use not only a familiar implementation but one that is a proven methodology, and one that can provide significant software reuse as they migrate from PCI to PCI Express.

This paper outlines how multiprocessor PCI Express systems will be implemented using industry standard practices established in the PCI paradigm. We first, however, will define the different usage models, and review the successful efforts in the PCI community to develop mechanisms to accommodate these requirements. Finally, we will cover how PCI Express systems will utilize non-transparent bridging to provide the functionality needed for these types of systems.

Usage Models

Intelligent Adapters

Intelligent adapters are typically peripheral devices that use a local processor to offload tasks from the host. Examples of intelligent adapters include RAID controllers, modem cards, and content processing blades that perform tasks such as security and flow processing. Generally, these tasks are either computationally onerous or require significant I/O bandwidth if performed by the host. By adding a local processor to the endpoint, system designers can enjoy significant incremental performance. In the RAID market, a significant number of products utilize local intelligence for their I/O processing.

Another example of intelligent adapters is an ecommerce blade. Because general purpose host processors are not optimized for the exponential mathematics necessary for SSL, utilizing a host processor to perform an SSL handshake typically reduces system performance by over 90%. Furthermore, one of the requirements for the SSL handshake operation is a true random number generator. Many general purpose processors do not have this feature, so it is actually difficult to perform SSL handshakes without dedicated hardware. Similar examples abound throughout the intelligent adapter marketplace; in fact, this usage model is so prevalent that for many applications it has become the de facto standard implementation.

Host Failover

Host failover capabilities are designed into systems that require high availability. High availability has become an increasingly important requirement, especially in storage and communication platforms. The only practical way to ensure that the overall system remains operational is to provide redundancy for all components. Host failover systems typically include a host based system attached to several endpoints. In addition, a backup host is attached to the system and is configured to monitor the system status. When the primary host fails, the backup host processor must not only recognize the failure, but then take steps to assume primary control, remove the failed host to prevent additional disruptions, reconstitute the system state, and continue the operation of the system without losing any data.

Multiprocessor Systems

Multiprocessor systems provide greater processing bandwidth by allowing multiple computational engines to simultaneously work on sections of a complex problem. Unlike systems utilizing host failover, where the backup processor is essentially idle, multiprocessor systems utilize all the engines to boost computational throughput. This enables a system to reach performance levels not possible by using only a single host processor. Multiprocessor systems typically consist of two or more complete sub-systems that can pass data between themselves via a special interconnect. A good example of a multihost system is a blade server chassis. Each blade is a complete subsystem, often replete with its own CPU, Direct Attached Storage, and I/O.

The History Multi-Processor Implementations Using PCI

To better understand the implementation proposed for PCI Express, one needs to first understand the PCI implementation.

PCI was originally defined in 1992 for personal computers. Because of the nature of PCs at that time, the protocol architects did not anticipate the need for multiprocessors. Therefore, they designed the system assuming that the host processor would enumerate the entire memory space. Obviously, if another processor is added, the system operation would fail as both processors would attempt to service the system requests.

1Several methodologies were subsequently invented to accommodate the requirement for multiprocessor capabilities using PCI. The most popular implementation, and the one discussed in this paper for PCI Express, is the use of non-transparent bridging between the processing subsystems to isolate their memory spaces.[1]

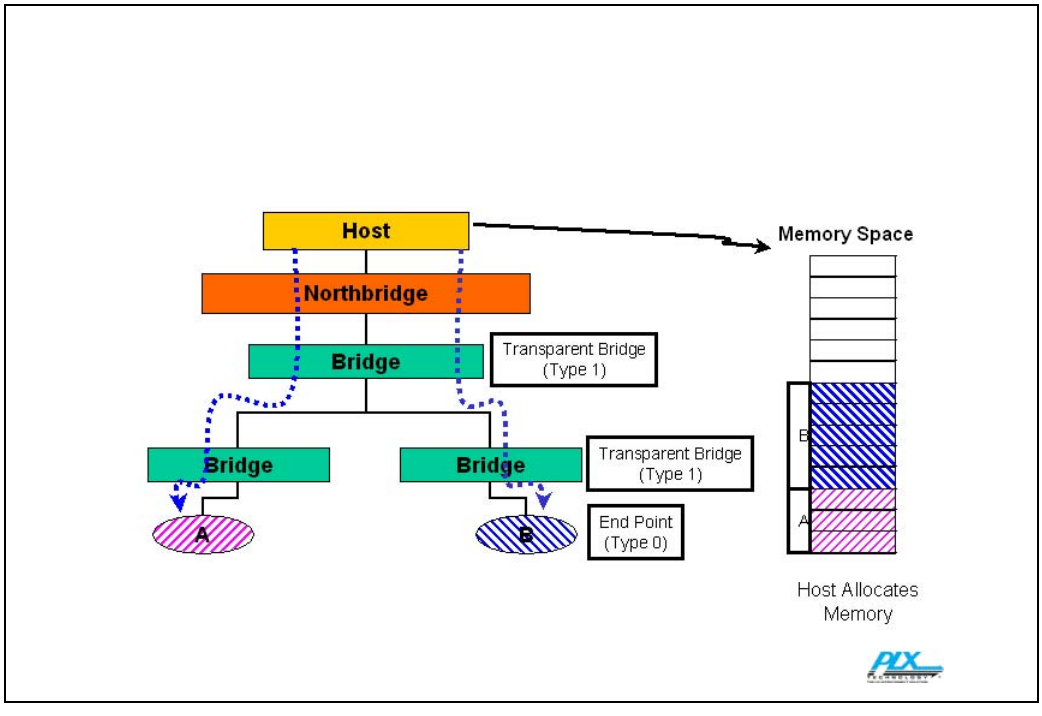
Because the host does not know the system topology when it is first powered up or reset, it must perform discovery to learn what devices are present and then map them into the memory space. To support standard discovery and configuration software, the PCI specification defines a standard format for Control and Status Registers (CSRs) of compliant devices. The standard PCI-to-PCI bridge CSR header, called a Type 1 header, includes primary, secondary and subordi-

1. Unless explicitly noted, the architecture for multiprocessor systems using PCI and PCI Express are similar and may be used interchangeably.

nate bus number registers that, when written by the host, define the CSR addresses of devices on the other side of the bridge. Bridges that employ a Type 1 CSR header are called transparent bridges.

A Type 0 header is used for endpoints. A Type 0 CSR header includes base address registers (BARs) used to request memory or I/O apertures from the host. Both Type 1 and Type 0 headers include a class code register that indicates what kind of bridge or endpoint is represented, with further information available in a subclass field and in device ID and vendor ID registers. The CSR header format and addressing rules allow the processor to search all the branches of a PCI hierarchy, from the host bridge down to each of its leaves, reading the class code registers of each device it finds as it proceeds, and assigning bus numbers as appropriate as it discovers PCI-to-PCI bridges along the way. At the completion of discovery, the host knows which devices are present and the memory and I/O space each device requires to function. These concepts are illustrated in Figure C - 1.

Figure C-1: Enumeration Using Transparent Bridges



Implementing Multi-host/Intelligent Adapters in PCI Express Base Systems

Up to this point, our discussions have been limited to one processor with one memory space. As technology progressed, system designers began developing end points with their own native processors built in. The problem that this caused was that both the host processor and the intelligent adapter would, upon power up or reset, attempt to enumerate the entire system, causing system conflict and ultimately a non-functional system.[2]

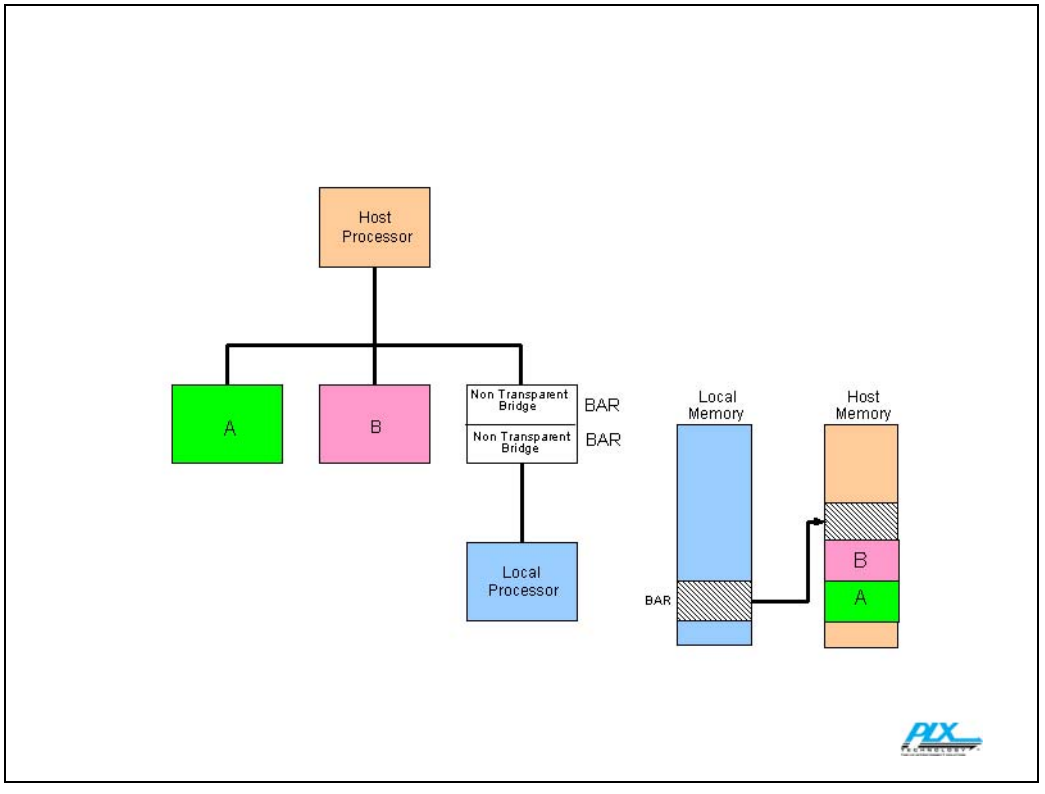
To get around this, architects designed non-transparent bridges. A non-transparent PCI-to-PCI Bridge, or PCI Express-to-PCI Express Bridge, is a bridge that exposes a Type 0 CSR header on both sides and forwards transactions from one side to the other with address translation, through apertures created by the BARs of those CSR headers. Because it exposes a Type 0 CSR header, the bridge appears to be an endpoint to discovery and configuration software, eliminating potential discovery software conflicts. Each BAR on each side of the bridge creates a tunnel or window into the memory space on the other side of the bridge. To facilitate communication between the processing domains on each side, the non-transparent bridge also typically includes doorbell registers to send interrupts from each side of the bridge to the other, and scratchpad registers accessible from both sides.

A non-transparent bridge is functionally similar to a transparent bridge in that both provide a path between two independent PCI buses (or PCI Express links). The key difference is that when a non-transparent bridge is used, devices on the downstream side of the bridge (relative to the system host) are not visible from the upstream side. This allows an intelligent controller on the downstream side to manage the devices in its local domain, while at the same time making them appear as a single device to the upstream controller. The path between the two buses allows the devices on the downstream side to transfer data directly to the upstream side of the bus without directly involving the intelligent controller in the data movement. Thus transactions are forwarded across the bus unfettered just as in a PCI-to-PCI Bridge, but the resources responsible are hidden from the host, which sees a single device.

2. While we are using an intelligent endpoint as the examples, we should note that a similar problem exists for multi-host systems.

Because we now have two memory spaces, the PCI Express system needs to translate addresses of transactions that cross from one memory space to the other. This is accomplished via Translation and Limit Registers associated with the BAR. See "Address Translation" on page 1013 for a detailed description; Figure C-2 on page 1004 provides a conceptual rendering of Direct Address Translation. Address translation can be done by Direct Address Translation (essentially replacement of the data under a mask), table lookup, or by adding an offset to an address. Figure C-3 on page 1005 shows Table Lookup Translation used to create multiple windows spread across system memory space for packet originated in a local I/O processor's domain, as well as Direct Address Translation used to create a single window in the opposite direction.

Figure C-2: Direct Address Translation



Appendix D
Class Codes

This appendix lists the class codes, sub-class codes, and programming interface byte definitions currently provided in the 2.3 PCI specification.

Figure D-1: Class Code Register

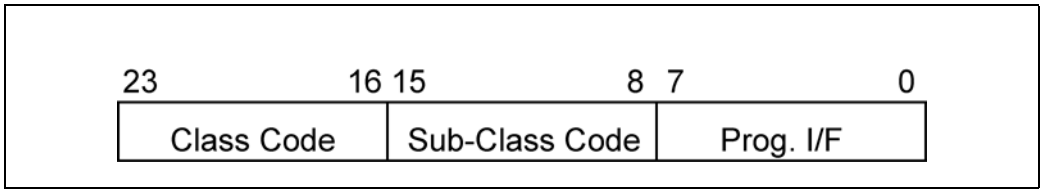


Table D-1: Defined Class Codes

Class	Description
00h	Function built before class codes were defined (in other words: before rev 2.0 of the PCI spec).
01h	Mass storage controller.
02h	Network controller.
03h	Display controller.
04h	Multimedia device.
05h	Memory controller.
06h	Bridge device.

Table D-1: Defined Class Codes (Continued)

Class	Description
07h	Simple communications controllers.
08h	Base system peripherals.
09h	Input devices.
0Ah	Docking stations.
0Bh	Processors.
0Ch	Serial bus controllers.
0Dh	Wireless controllers.
0Eh	Intelligent IO controllers.
0Fh	Satellite communications controllers.
10h	Encryption/Decryption controllers.
11h	Data acquisition and signal processing controllers.
12h-FEh	Reserved.
FFh	Device does not fit any of the defined class codes.

Table D-2: Class Code 0 (PCI rev 1.0)

Sub-Class	Prog. I/F	Description
00h	00h	All devices other than VGA.
01h	01h	VGA-compatible device.

Table D-3: Class Code 1: Mass Storage Controllers

Sub-Class	Prog. I/F	Description
00h	00h	SCSI controller.
01h	xxh	IDE controller. See Table D-20 on page 1031 for definition of Programming Interface byte.

Table D-3: Class Code 1: Mass Storage Controllers (Continued)

Sub-Class	Prog. I/F	Description
02h	00h	Floppy disk controller.
03h	00h	IPI controller.
04h	00h	RAID controller.
05h	20h	ATA controller with single DMA .
	30h	ATA controller with chained DMA.
80h	00h	Other mass storage controller.

Table D-4: Class Code 2: Network Controllers

Sub-Class	Prog. I/F	Description
00h	00h	Ethernet controller.
01h	00h	Token ring controller.
02h	00h	FDDI controller.
03h	00h	ATM controller.
04h	00h	ISDN Controller.
05h	00h	WorldFip controller.
06h		PICMG 2.14 Multi Computing. For information on the use of the Programming Interface Byte, see the PICMG 2.14 Multi Computing Specification (http://www.picmg.com).
80h	00h	Other network controller.

Table D-5: Class Code 3: Display Controllers

Sub-Class	Prog. I/F	Description
00h	00h	VGA-compatible controller, responding to memory addresses 000A0000h through 000BFFFFh (Video Frame Buffer), and IO addresses 03B0h through 3BBh, and 03C0h-through-03DFh and all aliases of these addresses.
	01h	8514-compatible controller, responding to IO address 02E8h and its aliases, 02EAh and 02EFh.
01h	00h	XGA controller.
02h	00h	3D Controller.
80h	00h	Other display controller.

Table D-6: Class Code 4: Multimedia Devices

Sub-Class	Prog. I/F	Description
00h	00h	Video device.
01h	00h	Audio device.
02h	00h	Computer Telephony device.
80h	00h	Other multimedia device.

Table D-7: Class Code 5: Memory Controllers

Sub-Class	Prog. I/F	Description
00h	00h	RAM memory controller.
01h	00h	Flash memory controller.
80h	00h	Other memory controller.

Table D-8: Class Code 6: Bridge Devices

Sub-Class	Prog. I/F	Description
00h	00h	Host/PCI bridge.
01h	00h	PCI/ISA bridge.
02h	00h	PCI/EISA bridge.
03h	00h	PCI/Micro Channel bridge.
04h	00h	PCI/PCI bridge.
	01h	Subtractive decode PCI-to-PCI bridge. Supports subtractive decode in addition to normal PCI-to-PCI functions. For a detailed discussion of this bridge type, refer to the MindShare *PCI System Architecture* book, Fourth Edition (published by Addison-Wesley).
05h	00h	PCI/PCMCIA bridge.
06h	00h	PCI/NuBus bridge.
07h	00h	PCI/CardBus bridge.
08h	xxh	RACEway bridge. RACEway is an ANSI standard (ANSI/VITA 5-1994) switching fabric. Bits 7:1 of the Interface bits are reserved, read-only and return zeros. Bit 0 is read-only and, if 0, indicates that the bridge is in Transparent mode, while 1 indicates that it's in End-Point mode.
09h	40h	Semi-transparent PCI-to-PCI bridge with the primary PCI bus side facing the system host processor.
	80h	Semi-transparent PCI-to-PCI bridge with the secondary PCI bus side facing the system host processor.
0Ah	00h	InfiniBand-to-PCI host bridge.
80h	00h	Other bridge type.

Table D-9: Class Code 7: Simple Communications Controllers

Sub-Class	Prog. I/F	Description
00h	00h	Generic XT-compatible serial controller.
	01h	16450-compatible serial controller.
	02h	16550-compatible serial controller.
	03h	16650-compatible serial controller.
	04h	16750-compatible serial controller.
	05h	16850-compatible serial controller.
	06h	16950-compatible serial controller.
01h	00h	Parallel port.
	01h	Bi-directional parallel port.
	02h	ECP 1.X-compliant parallel port.
	03h	IEEE 1284 controller.
	FEh	IEEE 1284 target device (not a controller).
02h	00h	Multiport serial controller.

Appendix E

Locked Transactions Series

Introduction

Native PCI Express implementations do not support lock. Support for Locked transaction sequences exist solely for supporting legacy device software executing on the host processor that performs a locked RMW (read-modify-write) operation on a memory semaphore that may reside within the memory of a legacy PCI device. This chapter defines the protocol defined by PCI Express for supporting locked access sequences that target legacy devices. Failure to support lock may result in deadlocks.

Background

PCI Express continues the PCI 2.3 tradition of supporting locked transaction sequences (RMW—ready-modify-write) to support legacy device software. PCI Express devices and their software drivers are never allowed to use instructions that cause the CPU to generate locked operations that target memory that resides beneath the Roor Complex level.

Locked operations consist of the basic RMW sequence, that is:

1. One or more memory reads from the target location to obtain the semaphore value.
2. The modification of the data within a processor register.
3. One or more writes to write the modified semaphore value back to the target memory location.

This transaction sequence must be performed such that no other accesses are permitted to the target locations (or device) during the locked sequence. This requires blocking other transactions during the operation. The result potentially can result in deadlocks and poor performance.

The devices required to support locked sequences are:

- The Root Complex.
- Any Switches in the path leading to a legacy devices that may be the target of a locked transaction series.
- A PCI Express - to - PCI Bridge.
- A PCI Express-to-PCI-X Bridge.
- Any legacy devices whose device drivers issue locked transactions to memory residing within the legacy device.

No other devices must support locked transactions and must ignore any locked transactions that they receive.

Lock in the PCI environment is achieved, in part, via the use of the PCI LOCK# signal. The equivalent functionality in PCI Express is accomplished via a transaction that emulates the LOCK signal functionality.

The PCI Express Lock Protocol

The only source of lock supported by PCI Express is the system processor, and, as a consequence, the source of all locked operations in PCI Express is the Root Complex (acting as the processor's surrogate). A locked operation is performed between a Root Complex downstream port and the PCI Express downstream port to which the targeted legacy device is attached. In most systems, the legacy device is typically a PCI Express-to-PCI or PCI Express-to-PCI-X bridge. Only one locked sequence at a time is supported for a given hierarchical path.

PCI Express limits locked transactions to Traffic Class 0 and Virtual Channel 0. All transactions with TC values other than zero that are mapped to a VC other than zero are permitted to traverse the fabric without regard to the locked operation. All transactions that are mapped to VC0 are subject to the lock rules described in this appendix. The discussion of the locked protocol in this appendix presumes that all transactions have been assigned to TC0 (unless otherwise indicated).

Lock Messages — The Virtual Lock Signal

PCI Express defines the following transactions that, together, act as a virtual wire that replaces the PCI LOCK# signal.

- **Memory Read Lock Request** (MRdLk) — Originates a locked sequence. The first MRdLk transaction blocks other requests from reaching the target device. One or more of these locked read requests may be issued during the sequence.
- **Memory Read Lock Completion with Data** (CplDLk) — Returns data and confirms that the path to the target is locked. A successful read Completion that returns data for the first Memory Read Lock request results in the path between the Root Complex and the target device being locked. That is, transactions traversing the same path from other ports are blocked from reaching either the root port or the target port. Transactions being routed in buffers for VC1-VC7 are unaffected by the lock.
- **Memory Read Lock Completion without Data** (CplLK) — A Completion without a data payload indicates that the lock sequence cannot complete currently and the path remains unlocked.
- **Unlock Message** — An unlock message is issued by the Root Complex from the locked root port. This message unlocks the path between the root port and the target port.

The Lock Protocol Sequence — an Example

This section explains the PCI Express lock protocol by example. The example includes the following devices:

- The Root Complex that initiates the Locked transaction series on behalf of the host processor.
- A Switch in the path between the root port and targeted legacy endpoint.
- A PCI Express-to-PCI Bridge in the path to the target.
- The target PCI device who's Device Driver initiated the locked RMW.
- A PCI Express endpoint is included to describe Switch behavior during lock.

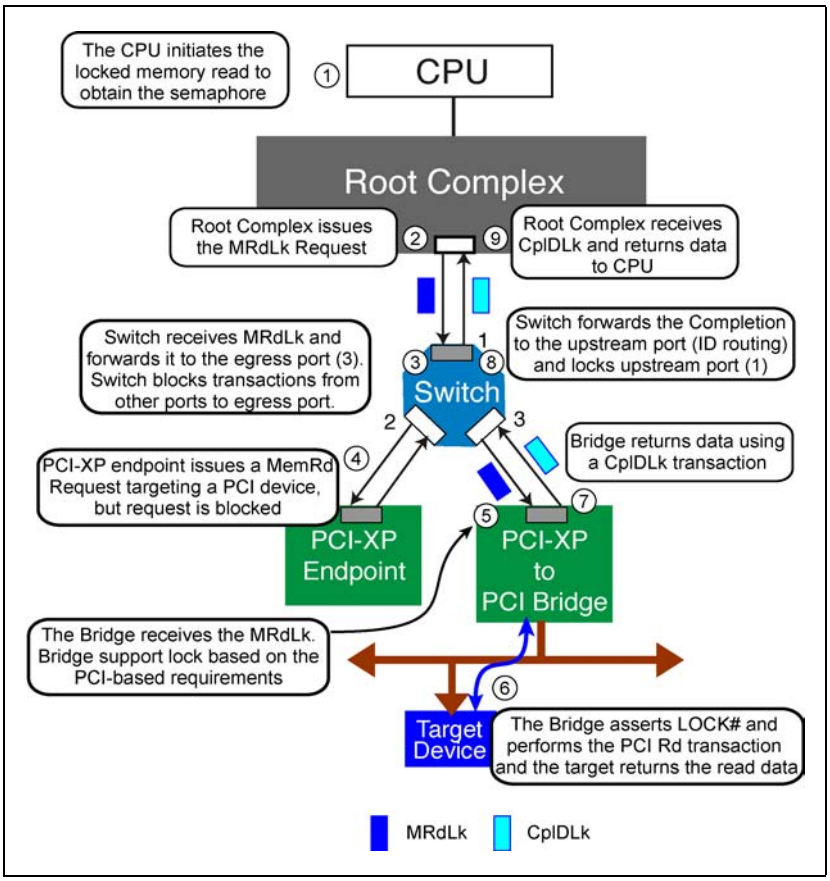
In this example, the locked operation completes normally. The steps that occur during the operation are described in the two sections that follow.

The Memory Read Lock Operation

Figure E-1 on page 1037 illustrates the first step in the Locked transaction series (i.e., the initial memory read to obtain the semaphore):

1. The CPU initiates the locked sequence (a Locked Memory Read) as a result of a driver executing a locked RMW instruction that targets a PCI target.
2. The Root Port issues a Memory Read Lock Request from port 2. The Root Complex is always the source of a locked sequence.
3. The Switch receives the lock request on its upstream port and forwards the request to the target egress port (3). The switch, upon forwarding the request to the egress port, must block all requests from ports other than the ingress port (1) from being sent from the egress port.
4. A subsequent peer-to-peer transfer from the illustrated PCI Express endpoint to the PCI bus (switch port 2 to switch port 3) would be blocked until the lock is cleared. Note that the lock is not yet established in the other direction. Transactions from the PCI Express endpoint could be sent to the Root Complex.
5. The Memory Read Lock Request is sent from the Switch's egress port to the PCI Express-to-PCI Bridge. This bridge will implement PCI lock semantics (See the MindShare book entitled *PCI System Architecture, Fourth Edition*, for details regarding PCI lock).
6. The bridge performs the Memory Read transaction on the PCI bus with the PCI LOCK# signal asserted. The target memory device returns the requested semaphore data to the bridge.
7. Read data is returned to the Bridge and is delivered back to the Switch via a Memory Read Lock Completion with Data (CplDLk).
8. The switch uses ID routing to return the packet upstream towards the host processor. When the CplDLk packet is forwarded to the upstream port of the Switch, it establishes a lock in the upstream direction to prevent traffic from other ports from being routed upstream. The PCI Express endpoint is completely blocked from sending any transaction to the Switch ports via the path of the locked operation. Note that transfers between Switch ports not involved in the locked operation would be permitted (not shown in this example).
9. Upon detecting the CplDLk packet, the Root Complex knows that the lock has been established along the path between it and the target device, and the completion data is sent to the CPU.

Figure E-1: Lock Sequence Begins with Memory Read Lock Request



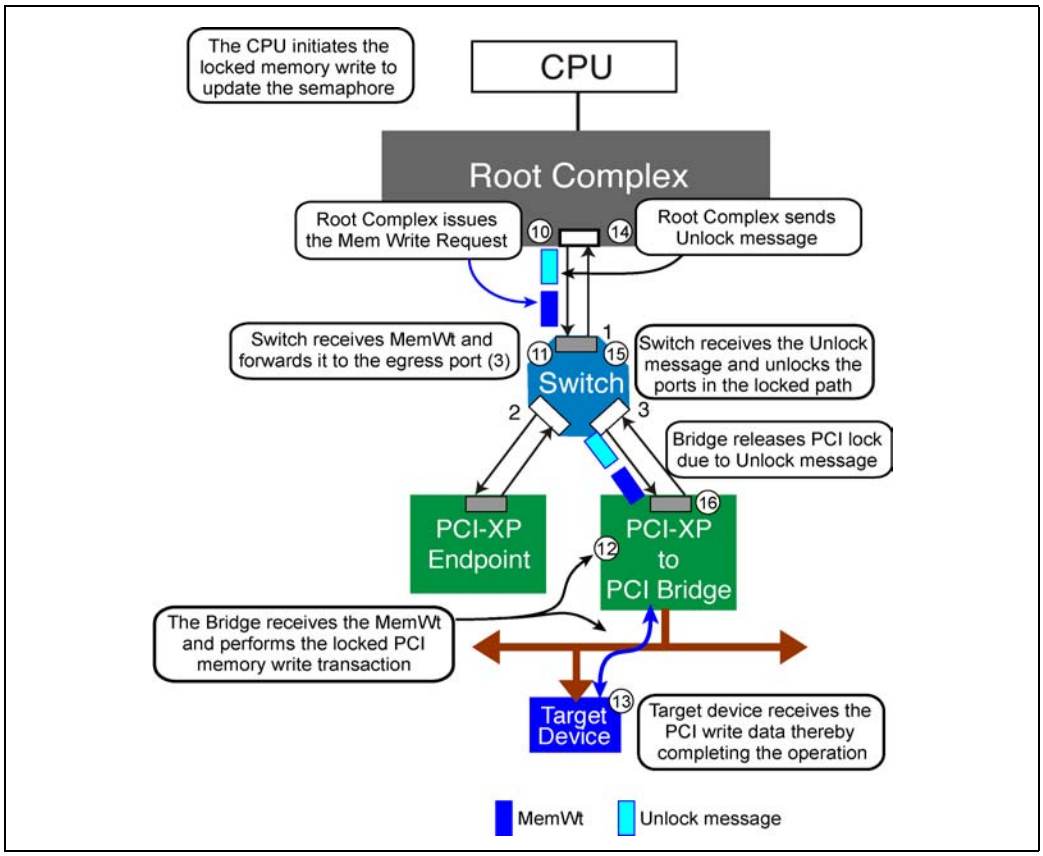
Read Data Modified and Written to Target and Lock Completes

The device driver receives the semaphore value, alters it, and then initiates a memory write to update the semaphore within the memory of the legacy PCI device. Figure E-2 on page 1038 illustrates the write sequence followed by the Root Complex's transmission of the Unlock message that releases the lock:

10. The Root Complex issues the Memory Write Request across the locked path to the target device.
11. The Switch forwards the transaction to the target egress port (3). The memory address of the Memory Write must be the same as the initial Memory Read request.

12. The bridge forwards the transaction to the PCI bus.
13. The target device receives the memory write data.
14. Once the Memory Write transaction is sent from the Root Complex, it sends an Unlock message to instruct the Switches and any PCI/PCI-X bridges in the locked path to release the lock. Note that the Root Complex presumes the operation has completed normally (because memory writes are posted and no Completion is returned to verify success).
15. The Switch receives the Unlock message, unlocks its ports and forwards the message to the egress port that was locked to notify any other Switches and/or bridges in the locked path that the lock must be cleared.
16. Upon detecting the Unlock message, the bridge must also release the lock on the PCI bus.

Figure E-2: Lock Completes with Memory Write Followed by Unlock Message



PC Programming/Hardware

"*PCI Express System Architecture* is a high quality and comprehensive must-have reference for any engineer working with PCI Express. Highly recommended."

—David Churchill | Agilent Technologies

PCI Express is the third-generation Peripheral Component Interconnect technology for a wide range of systems and peripheral devices. Incorporating recent advances in high-speed, point-to-point interconnects, PCI Express provides significantly higher performance, reliability, and enhanced capabilities—at a lower cost—than the previous PCI and PCI-X standards. Therefore, anyone working on next-generation PC systems, BIOS and device driver development, and peripheral device design will need to have a thorough understanding of PCI Express.

PCI Express System Architecture provides an in-depth description and comprehensive reference to the PCI Express standard. The book contains information needed for design, verification, and test, as well as background information essential for writing low-level BIOS and device drivers. In addition, it offers valuable insight into the technology's evolution and cutting-edge features.

Following an overview of the PCI Express architecture, the book moves on to cover transaction protocols, the physical/electrical layer, power management, configuration, and more. Specific topics covered include:

- Split transaction protocol
- Packet format and definition, including use of each field
- ACK/NAK protocol
- Traffic Class and Virtual Channel applications and use
- Flow control initialization and operation
- Error checking mechanisms and reporting options
- Switch design issues
- Advanced Power Management mechanisms and use
- Active State Link power management
- Hot Plug design and operation
- Message transactions
- Physical layer functions
- Electrical signaling characteristics and issues
- PCI Express enumeration procedures
- Configuration register definitions

Thoughtfully organized, featuring a plethora of illustrations, and comprehensive in scope, ***PCI Express System Architecture*** is an essential resource for anyone working with this important technology.

MindShare's PC System Architecture Series is a crisply written and comprehensive set of guides to the most important PC hardware standards. Books in the series are intended for use by hardware and software designers, programmers, and support personnel.

MindShare, Inc., is one of the leading technical training companies in the hardware industry, providing innovative courses for dozens of companies, including IBM, HP, PLX, Sun, and Texas Instruments.

Ravi Budruk is a senior staff engineer and instructor with MindShare, Inc., where he has trained hundreds of engineers. He is an industry expert on such topics as Intel Processor and PC architecture, as well as such bus architectures as PCI Express, PCI, PCI-X, HyperTransport, IEEE 1394, and ISA. Before working at MindShare, Mr. Budruk was a PC chipset architect and designer at VLSI Technology, Inc.

Don Anderson is an expert on digital electronics and system design. He passes on his wealth of experience by training engineers, programmers, and technicians at MindShare, Inc., and is the author of numerous MindShare books.

Tom Shanley is President of MindShare, Inc., and one of the world's foremost authorities on computer system architecture.

www.informit.com/aw
www.mindshare.com

Cover design by Barbara T. Atkinson
Cover photograph by Tatsuhiko Shimada/Photonica

Contact www.mindshare.com for Training on This Subject

ISBN-13: 978-0-321-15630-3
ISBN-10: 0-321-15630-7
57699
9 780321 156303
$76.99 U.S./$84.99 CANADA

Text printed on recycled paper
Addison-Wesley
Pearson Education