"MindShare books are critical in the understanding of complex technical topics, such as PCI Express 3.0 architecture. Many of our customers and industry partners depend on these books for the success of their projects"

Joe Mendolia - Vice President, LeCroy

MINDSHARE

BRINGING LIFE TO KNOWLEDGE

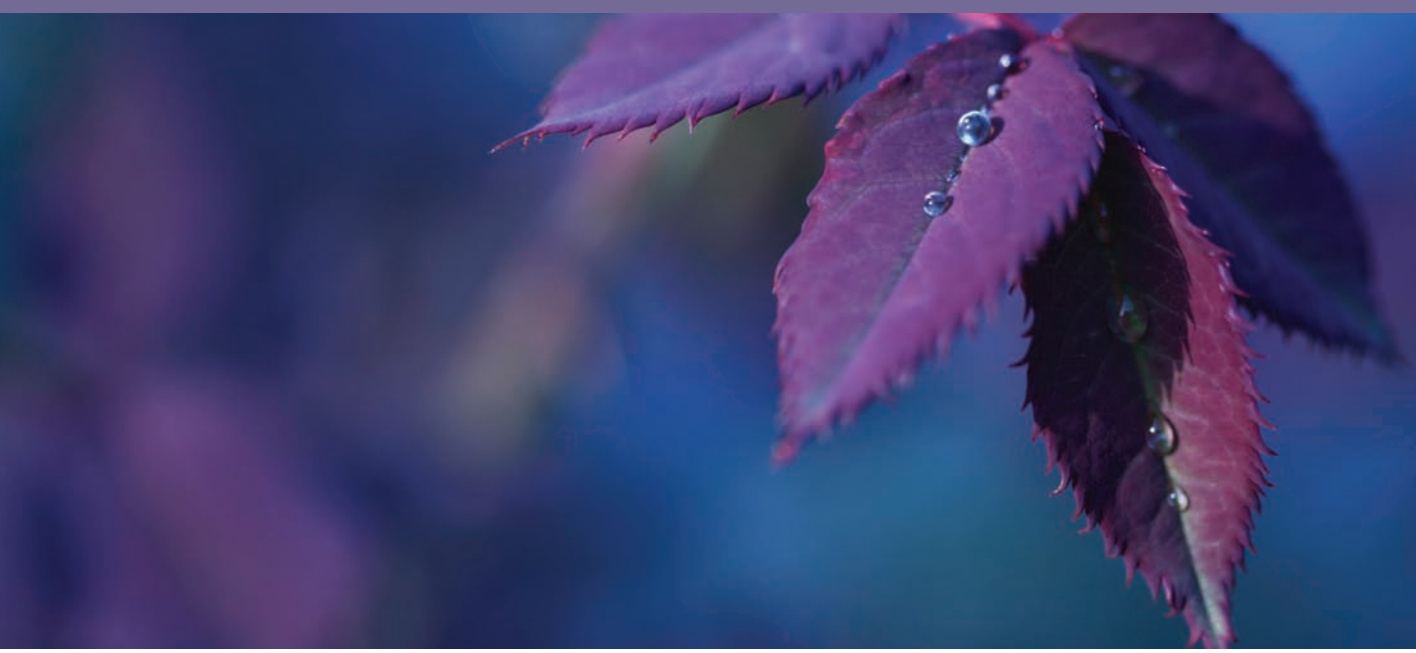
For training, visit mindshare.com

MindShare Technology Series

PCI Express Technology

Comprehensive Guide to Generations 1.x, 2.x and 3.0

Mike Jackson, Ravi Budruk | MindShare, Inc.



PCI Express Technology

Comprehensive Guide to Generations 1.x, 2.x, 3.0

MINDSHARE, INC.

Mike Jackson
Ravi Budruk
Technical Edit by Joe Winkles and Don Anderson

MindShare Live Training and Self-Paced Training

Intel Architecture • Intel Ivy Bridge Processor • Intel 64 (x86) Architecture • Intel QuickPath Interconnect (QPI) • Computer Architecture	**Virtualization Technology** • PC Virtualization • IO Virtualization
AMD Architecture • AMD Opteron Processor (Bulldozer) • AMD64 Architecture	**IO Buses** • PCI Express 3.0 • USB 3.0 / 2.0 • xHCI for USB
Firmware Technology • UEFI Architecture • BIOS Essentials	**Storage Technology** • SAS Architecture • Serial ATA Architecture • NVMe Architecture
ARM Architecture • ARM Architecture	**Memory Technology** • Modern DRAM Architecture
Graphics Architecture • Graphics Hardware Architecture	**High Speed Design** • High Speed Design • EMI/EMC
Programming • X86 Architecture Programming • X86 Assembly Language Basics • OpenCL Programming	**Surface-Mount Technology (SMT)** • SMT Manufacturing • SMT Testing

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

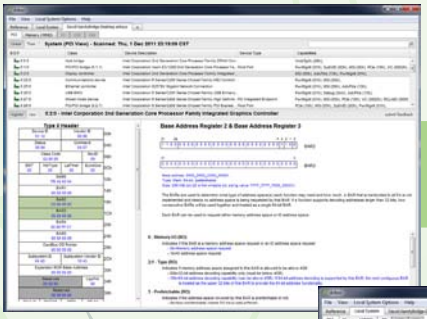
MindShare has over 25 years experience in conducting technical training on cutting-edge technologies. We understand the challenges companies have when searching for quality, effective training which reduces the students' time away from work and provides cost-effective alternatives. MindShare offers many 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.

MindShare offers numerous courses in a self-paced training format (eLearning). We've taken our 25+ years of experience in the technical training industry and made that knowledge available to you at the click of a mouse.

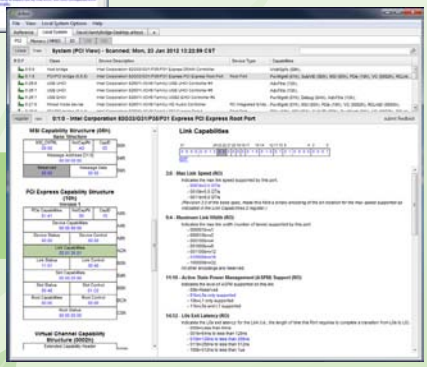
training@mindshare.com **1-800-633-1440** **www.mindshare.com**

ARBOR
BY MINDSHARE

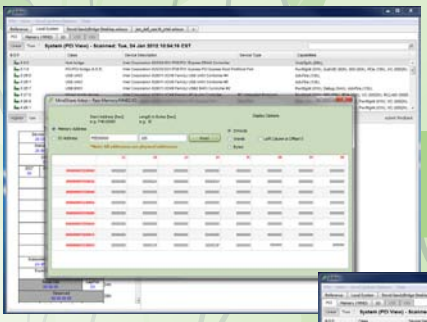
The Ultimate Tool to View, Edit and Verify Configuration Settings of a Computer



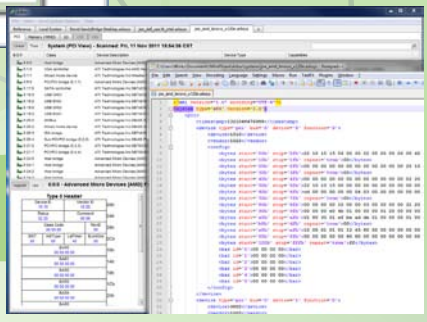
Decode Data from Live Systems



Apply Standard and Custom Rule Checks



Directly Edit Config, Memory and IO Space



Everything Driven from Open Format XML

Feature List

- Scan config space for all PCI-visible functions in system
- Run standard and custom rule checks to find errors and non-optimal settings
- Write to any config space location, memory address or IO address
- View standard and non-standard structures in a decoded format
- Import raw scan data from other tools (e.g. lspci) to view in Arbor's decoded format
- Decode info included for standard PCI, PCI-X and PCI Express structures
- Decode info included for some x86-based structures and device-specific registers
- Create decode files for structures in config space, memory address space and IO space
- Save system scans for viewing later or on other systems
- All decode files and saved system scans are XML-based and open-format

COMING SOON

Decoded view of x86 structures (MSRs, ACPI, Paging, Virtualization, etc.)

mindshare.com | 800.633.1440 | training@mindshare.com

ARBOR BY MINDSHARE

The Ultimate Tool to View, Edit and Verify Configuration Settings of a Computer

MindShare Arbor is a computer system debug, validation, analysis and learning tool that allows the user to read and write any memory, IO or configuration space address. The data from these address spaces can be viewed in a clean and informative style as well as checked for configuration errors and non-optimal settings.

View Reference Info

MindShare Arbor is an excellent reference tool to quickly look at standard PCI, PCI-X and PCIe structures. All the register and field definitions are up-to-date with the PCI Express 3.0. x86, ACPI and USB reference info will be coming soon as well.

Decoding Standard and Custom Structures from a Live System

MindShare Arbor can perform a scan of the system it is running on to record the config space from all PCI-visible functions and show it in a clean and intuitive decoded format. In addition to scanning PCI config space, MindShare Arbor can also be directed to read any memory address space and IO address space and display the collected data in the same decoded fashion.

Run Rule Checks of Standard and Custom Structures

In addition to capturing and displaying headers and capability structures from PCI config space, Arbor can also check the settings of each field for errors (e.g. violates the spec) and non-optimal values (e.g. a PCIe link trained to something less than its max capability). MindShare Arbor has scores of these checks built in and can be run on any system scan (live or saved). Any errors or warnings are flagged and displayed for easy evaluation and debugging.
MindShare Arbor allows users to create their own rule checks to be applied to system scans. These rule checks can be for any structure, or set of structures, in PCI config space, memory space or IO space. The rule checks are written in JavaScript. (Python support coming soon.)

Write Capability

MindShare Arbor provides a very simple interface to directly edit a register in PCI config space, memory address space or IO address space. This can be done in the decoded view so you see what the meaning of each bit, or by simply writing a hex value to the target location.

Saving System Scans (XML)

After a system scan has been performed, MindShare Arbor allows saving of that system's scanned data (PCI config space, memory space and IO space) all in a single file to be looked at later or sent to a colleague. The scanned data in these Arbor system scan files (.ARBSYS files) are XML-based and can be looked at with any text editor or web browser. Even scans performed with other tools can be easily converted to the Arbor XML format and evaluated with MindShare Arbor.

1 Background

This Chapter

This chapter reviews the PCI (Peripheral Component Interface) bus models that preceded PCI Express (PCIe) as a way of building a foundation for understanding PCI Express architecture. PCI and PCI-X (PCI-eXtended) are introduced and their basic features and characteristics are described, followed by a discussion of the motivation for migrating from those earlier parallel bus models to the serial bus model used by PCIe.

The Next Chapter

The next chapter provides an introduction to the PCI Express architecture and is intended to serve as an "executive level" overview, covering all the basics of the architecture at a high level. It introduces the layered approach to PCIe port design given in the spec and describes the responsibilities of each layer.

Introduction

Establishing a solid foundation in the technologies on which PCIe is built is a helpful first step to understanding it, and an overview of those architectures is presented here. Readers already familiar with PCI may prefer to skip to the next chapter. This background is only intended as a brief overview. For more depth and detail on PCI and PCI-X, please refer to MindShare's books: PCI System Architecture, and PCI-X System Architecture.

As an example of how this background can be helpful, the software used for PCIe remains much the same as it was for PCI. Maintaining this backward compatibility encourages migration from the older designs to the new by making the software changes as simple and inexpensive as possible. As a result, older PCI software works unchanged in a PCIe system and new software will continue to use the same models of operation. For this reason and others, understanding PCI and its models of operation will facilitate an understanding of PCIe.

PCI and PCI-X

The PCI (Peripheral Component Interface) bus was developed in the early 1990's to address the shortcomings of the peripheral buses that were used in PCs (personal computers) at the time. The standard at the time was IBM's AT (Advanced Technology) bus, referred to by other vendors as the ISA (Industry Standard Architecture) bus. ISA had been sufficient for the 286 16-bit machines for which it was designed, but additional bandwidth and improved capabilities, such plug-and-play, were needed for the newer 32-bit machines and their peripherals. Besides that, ISA used big connectors that had a high pin count. PC vendors recognized the need for a change and several alternate bus designs were proposed, such as IBM's MCA (Micro-Channel Architecture), the EISA bus (Extended ISA, proposed as an open standard by IBM competitors), and the VESA bus (Video Electronics Standards Association, proposed by video card vendors for video devices). However, all of these designs had drawbacks that prevented wide acceptance. Eventually, PCI was developed as an open standard by a consortium of major players in the PC market who formed a group called the PCISIG (PCI Special Interest Group). The performance of the newly-developed bus architecture was much higher than ISA, and it also defined a new set of registers within each device referred to as configuration space. These registers allowed software to see the memory and IO resources a device needed and assign each device addresses that wouldn't conflict with other addresses in the system. These features: open design, high speed, and software visibility and control, helped PCI overcome the obstacles that had limited ISA and other buses PCI quickly became the standard peripheral bus in PCs.

A few years later, PCI-X (PCI-eXtended) was developed as a logical extension of the PCI architecture and improved the performance of the bus quite a bit. We'll discuss the changes a little later, but a major design goal for PCI-X was maintaining compatibility with PCI devices, both in hardware and software, to make migration from PCI as simple as possible. Later, the PCI-X 2.0 revision added even higher speeds, achieving a raw data rate of up to 4 GB/s. Since PCI-X maintained hardware backward compatibility with PCI, it remained a parallel bus and inherited the problems associated with that model. That's interesting for us because parallel buses eventually reach a practical ceiling on effective bandwidth and can't readily be made to go faster. Going to a higher data rate with PCI-X was explored by the PCISIG, but the effort was eventually abandoned. That speed ceiling, along with a high pin count, motivated the transition away from the parallel bus model to the new serial bus model.

These earlier bus definitions are listed in Table 1-1 on page 11, which shows the development over time of higher frequencies and bandwidths. One of the inter-

esting things to note in this table is the correlation of clock frequency and the number of add-in card slots on the bus. This was due to PCI's low-power signaling model, which meant that higher frequencies required shorter traces and fewer loads on the bus (see "Reflected-Wave Signaling" on page 16). Another point of interest is that, as the clock frequency increases, the number of devices permitted on the shared bus decreases. When PCI-X 2.0 was introduced, its high speed mandated that the bus become a point-to-point interconnect.

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

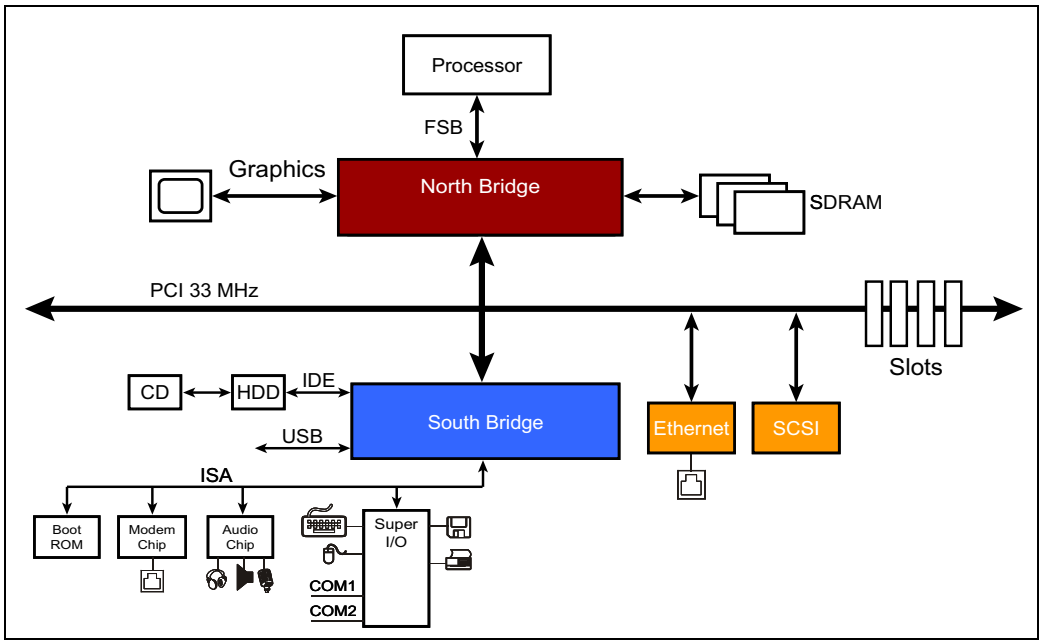
Bus Type	Clock Frequency	Peak Bandwidth 32-bit - 64-bit bus	Number of Card Slots per Bus
PCI	33 MHz	133 - 266 MB/s	4-5
PCI	66 MHz	266 - 533 MB/s	1-2
PCI-X 1.0	66 MHz	266 - 533 MB/s	4
PCI-X 1.0	133 MHz	533 - 1066 MB/s	1-2
PCI-X 2.0 (DDR)	133 MHz	1066 - 2132 MB/s	1 (point-to-point bus)
PCI-X 2.0 (QDR)	133 MHz	2132 - 4262 MB/s	1 (point-to-point bus)

PCI Basics

Basics of a PCI-Based System

Figure 1-1 on page 12 shows an older system based on a PCI bus. The system includes a North Bridge (called "north" because if the diagram is viewed as a map, it appears geographically north of the central PCI bus) that interfaces between the processor and the PCI bus. Associated with the North Bridge is the processor bus, system memory bus, AGP graphics bus, and PCI. Several devices share the PCI bus and are either connected directly to the bus or plugged into an add-in card connector. A South Bridge connects PCI to system peripherals, such as the ISA bus where legacy peripherals were carried forward for a few years. The South Bridge was typically also the central resource for PCI that provided system signals like reset, reference clock, and error reporting.

Figure 1-1: Legacy PCI Bus-Based Platform



PCI Bus Initiator and Target

In a PCI hierarchy each device on the bus may contain up to eight functions that all share the bus interface for that device, numbered 0-7 (a single-function device is always assigned function number 0). Every function is capable of acting as a target for transactions on the bus, and most will also be able to initiate transactions. Such an initiator (called a Bus Master) has a pair of pins (REQ# and GNT#) dedicated to arbitrating for use of the shared PCI bus. As shown in Figure 1-2 on page 13, a Request (REQ#) pin indicates that the master needs to use the bus and is sent to the bus arbiter for evaluation against all the other requests at that moment. The arbiter is often located in the bridge that is hierarchically above the bus and receives arbitration requests from all the devices that can act as initiators (Bus Masters) on that bus. The arbiter decides which requester should be the next owner of the bus and asserts the Grant (GNT#) pin for that device. According to the protocol, whenever the previous transaction finishes and the bus goes idle, whichever device sees its GNT# asserted at that time is designated as the next Bus Master and can begin its transaction.

2 PCIe Architecture Overview

Previous Chapter

The previous chapter provided historical background to establish a foundation for understanding PCI Express. This included reviewing the basics of PCI and PCI-X 1.0/2.0. The goal was to provide a context for the overview of PCI Express that follows.

This Chapter

This chapter provides a thorough introduction to the PCI Express architecture and is intended to serve as an "executive level" overview, covering all the basics of the architecture at a high level. It introduces the layered approach given in the spec and describes the responsibilities of each layer. The various packet types are introduced along with the protocol used to communicate them and facilitate reliable transmission.

The Next Chapter

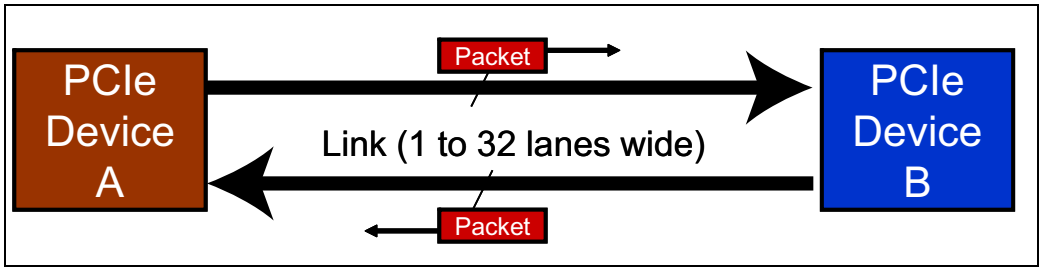
The next chapter provides an introduction to configuration in the PCI Express environment. This includes the space in which a Function's configuration registers are implemented, how a Function is discovered, how configuration transactions are generated and routed, the difference between PCI-compatible space and PCIe extended space, and how software differentiates between an Endpoint and a Bridge.

Introduction to PCI Express

PCI Express represents a major shift from the parallel bus model of its predecessors. As a serial bus, it has more in common with earlier serial designs like InfiniBand or Fibre Channel, but it remains fully backward compatible with PCI in software.

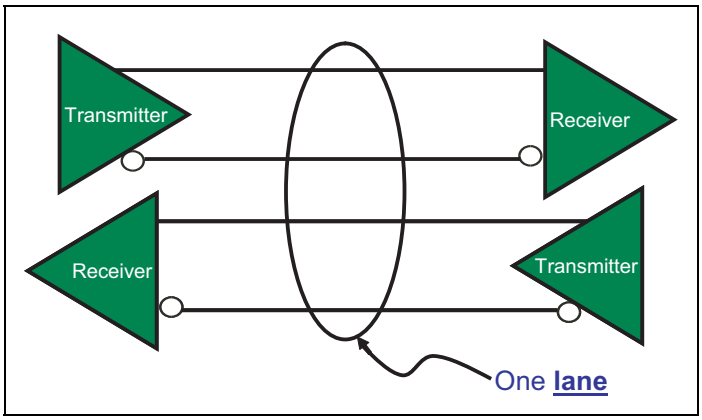
As is true of many high-speed serial transports, PCIe uses a bidirectional connection and is capable of sending and receiving information at the same time. The model used is referred to as a dual-simplex connection because each interface has a simplex transmit path and a simplex receive path, as shown in Figure 2-1 on page 40. Since traffic is allowed in both directions at once, the communication path between two devices is technically full duplex, but the spec uses the term dual-simplex because it's a little more descriptive of the actual communication channels that exist.

Figure 2-1: Dual-Simplex Link



The term for this path between the devices is a **Link**, and is made up of one or more transmit and receive pairs. One such pair is called a **Lane**, and the spec allows a Link to be made up 1, 2, 4, 8, 12, 16, or 32 Lanes. The number of lanes is called the Link Width and is represented as x1, x2, x4, x8, x16, and x32. The trade-off regarding the number of lanes to be used in a given design is straightforward: more lanes increase the bandwidth of the Link but add to its cost, space requirement, and power consumption. For more on this, see "Links and Lanes" on page 46.

Figure 2-2: One Lane



Software Backward Compatibility

One of the most important design goals for PCIe was backward compatibility with PCI software. Encouraging migration away from a design that is already installed and working in existing systems requires two things: First, a compelling improvement that motivates even considering a change and, second, minimizing the cost, risk, and effort of changing. A common way to help this second factor in computers is to maintain the viability of software written for the old model in the new one. To achieve this for PCIe, all the address spaces used for PCI are carried forward either unchanged or simply extended. Memory, IO, and Configuration spaces are still visible to software and programmed in exactly the same way they were before. Consequently, software written years ago for PCI (BIOS code, device drivers, etc.) will still work with PCIe devices today. The configuration space has been extended dramatically to include many new registers to support new functionality, but the old registers are still there and still accessible in the regular way (see "Software Compatibility Characteristics" on page 49).

Serial Transport

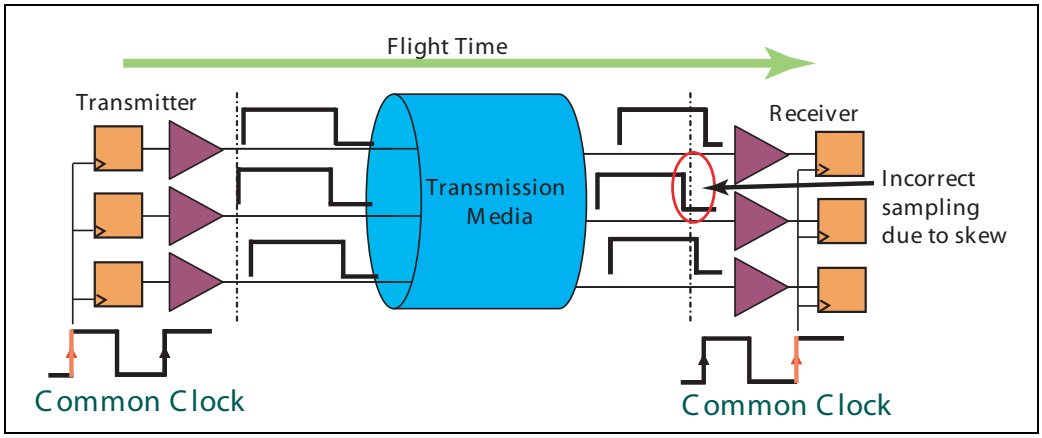
The Need for Speed

Of course, a serial model must run much faster than a parallel design to accomplish the same bandwidth because it may only send one bit at a time. This has not proven difficult, though, and in the past PCIe has worked reliably at 2.5 GT/s and 5.0 GT/s. The reason these and still higher speeds (8 GT/s) are attainable is that the serial model overcomes the shortcomings of the parallel model.

Overcoming Problems. By way of review, there are a handful of problems that limit the performance of a parallel bus and three are illustrated in Figure 2-3 on page 42. To get started, recall that parallel buses use a common clock; outputs are clocked out on one clock edge and clocked into the receiver on the next edge. One issue with this model is the time it takes to send a signal from transmitter to receiver, called the flight time. The flight time must be less than the clock period or the model won't work, so going to smaller clock periods is challenging. To make this possible, traces must get shorter and loads reduced but eventually this becomes impractical. Another factor is the difference in the arrival time of the clock at the sender and receiver, called clock skew. Board layout designers work hard to minimize this value because it detracts from the timing budget but it can never be eliminated. A third factor is signal skew, which is

the difference in arrival times for all the signals needed on a given clock. Clearly, the data can't be latched until all the bits are ready and stable, so we end up waiting for the slowest one.

Figure 2-3: Parallel Bus Limitations



How does a serial transport like PCIe get around these problems? First, flight time becomes a non-issue because the clock that will latch the data into the receiver is actually built into the data stream and no external reference clock is necessary. As a result, it doesn't matter how small the clock period is or how long it takes the signal to arrive at the receiver because the clock arrives with it at the same time. For the same reason there's no clock skew, again because the latching clock is recovered from the data stream. Finally, signal skew is eliminated within a Lane because there's only one data bit being sent. The signal skew problem returns if a multi-lane design is used, but the receiver corrects for this automatically and can fix a generous amount of skew. Although serial designs overcome many of the problems of parallel models, they have their own set of complications. Still, as we'll see later, the solutions are manageable and allow for high-speed, reliable communication.

Bandwidth. The combination of high speed and wide Links that PCIe supports can result in some impressive bandwidth numbers, as shown in Table 2-1 on page 43. These numbers are derived from the bit rate and bus characteristics. One such characteristic is that, like many other serial transports, the first two generations of PCIe use an encoding process called **8b/10b** that generates a 10-bit output based on an 8-bit input. In spite of the overhead this introduces, there are several good reasons for doing it as we'll see later. For now it's enough to

3 Configuration Overview

The Previous Chapter

The previous chapter provides a thorough introduction to the PCI Express architecture and is intended to serve as an "executive level" overview. It introduces the layered approach to PCIe port design described in the spec. The various packet types are introduced along with the transaction protocol.

This Chapter

This chapter provides an introduction to configuration in the PCIe environment. This includes the space in which a Function's configuration registers are implemented, how a Function is discovered, how configuration transactions are generated and routed, the difference between PCI-compatible configuration space and PCIe extended configuration space, and how software differentiates between an Endpoint and a Bridge.

The Next Chapter

The next chapter describes the purpose and methods of a function requesting memory or IO address space through Base Address Registers (BARs) and how software initializes them. The chapter describes how bridge Base/Limit registers are initialized, thus allowing switches to route TLPs through the PCIe fabric.

Definition of Bus, Device and Function

Just as in PCI, every PCIe Function is uniquely identified by the Device it resides within and the Bus to which the Device connects. This unique identifier is commonly referred to as a 'BDF'. Configuration software is responsible for detecting every Bus, Device and Function (BDF) within a given topology. The following sections discuss the primary BDF characteristics in the context of a sample PCIe topology. Figure 3-1 on page 87 depicts a PCIe topology that high-

lights the Buses, Devices and Functions implemented in a sample system. Later in this chapter the process of assigning Bus and Device Numbers is explained.

PCIe Buses

Up to 256 Bus Numbers can be assigned by configuration software. The initial Bus Number, Bus 0, is typically assigned by hardware to the Root Complex. Bus 0 consists of a Virtual PCI bus with integrated endpoints and Virtual PCI-to-PCI Bridges (P2P) which are hard-coded with a Device number and Function number. Each P2P bridge creates a new bus that additional PCIe devices can be connected to. Each bus must be assigned a unique bus number. Configuration software begins the process of assigning bus numbers by searching for bridges starting with Bus 0, Device 0, Function 0. When a bridge is found, software assigns the new bus a bus number that is unique and larger than the bus number the bridge lives on. Once the new bus has been assigned a bus number, software begins looking for bridges on the new bus before continuing scanning for more bridges on the current bus. This is referred to as a "depth first search" and is described in detail in "Enumeration - Discovering the Topology" on page 104.

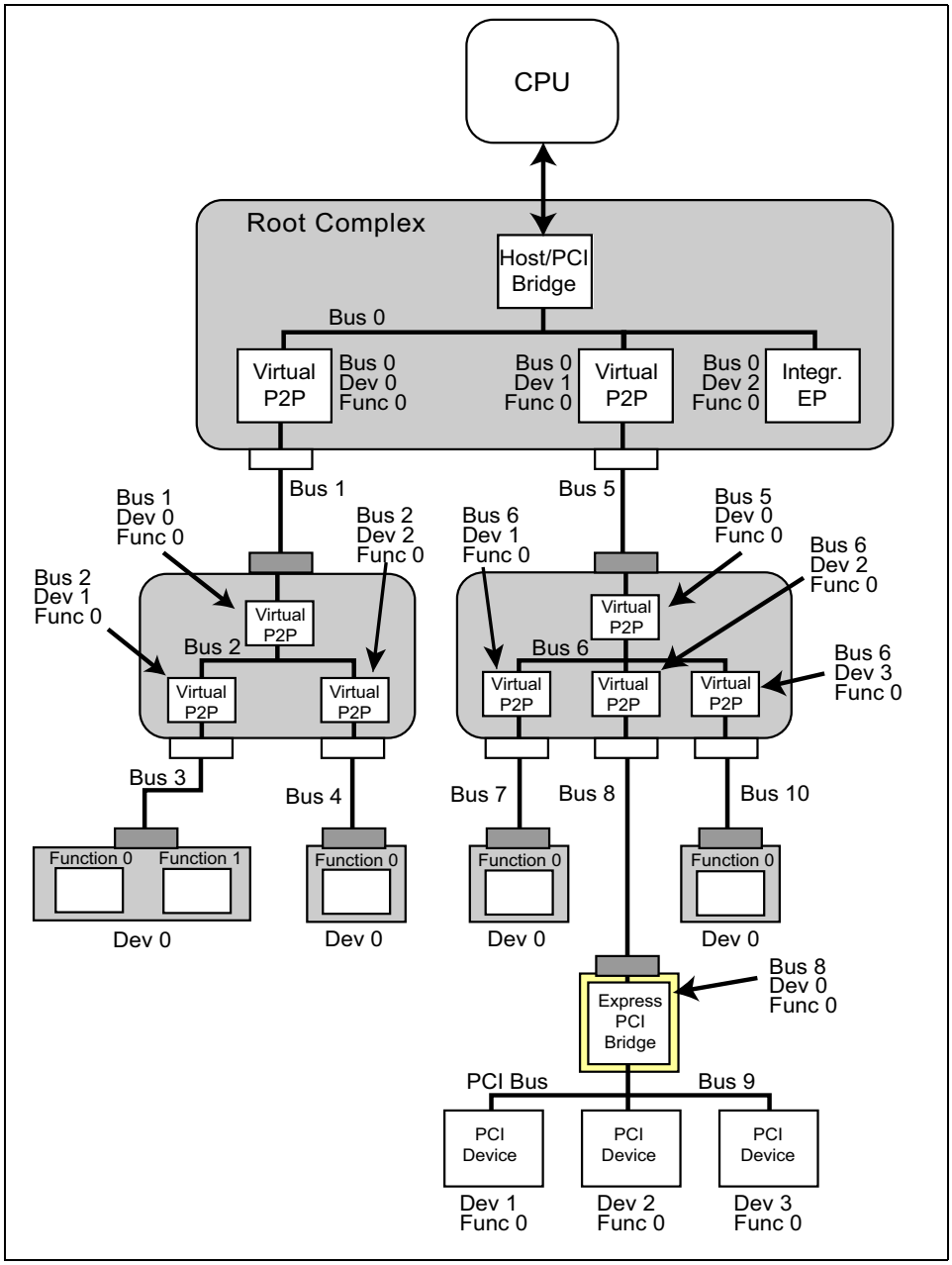
PCIe Devices

PCIe permits up to 32 device attachments on a single PCI bus, however, the point-to-point nature of PCIe means only a single device can be attached directly to a PCIe link and that device will always end up being Device 0. Root Complexes and Switches have Virtual PCI buses which do allow multiple Devices being "attached" to the bus. Each Device must implement Function 0 and may contain a collection of up to eight Functions. When two or more Functions are implemented the Device is called a multi-function device.

PCIe Functions

As previously discussed Functions are designed into every Device. These Functions may include hard drive interfaces, display controllers, ethernet controllers, USB controllers, etc. Devices that have multiple Functions do not need to be implemented sequentially. For example, a Device might implement Functions 0, 2, and 7. As a result, when configuration software detects a multifunction device, each of the possible Functions must be checked to learn which of them are present. Each Function also has its own configuration address space that is used to setup the resources associated with the Function.

Figure 3-1: Example System



Configuration Address Space

The first PCs required users to set switches and jumpers to assign resources for each card installed and this frequently resulted in conflicting memory, IO and interrupt settings. The subsequent IO architectures, Extended ISA (EISA) and the IBM PS2 systems, were the first to implemented plug and play architectures. In these architectures configuration files were shipped with each plug-in card that allowed system software to assign basic resources. PCI extended this capability by implementing standardized configuration registers that permit generic shrink-wrapped OSs to manage virtually all system resources. Having a standard way to enable error reporting, interrupt delivery, address mapping and more, allows one entity, the configuration software, to allocate and configure the system resources which virtually eliminates resource conflicts.

PCI defines a dedicated block of configuration address space for each Function. Registers mapped into the configuration space allow software to discover the existence of a Function, configure it for normal operation and check the status of the Function. Most of the basic functionality that needs to be standardized is in the header portion of the configuration register block, but the PCI architects realized that it would beneficial to standardize optional features, called capability structures (e.g. Power Management, Hot Plug, etc.). The PCI-Compatible configuration space includes 256 bytes for each Function.

PCI-Compatible Space

Refer to Figure 3-2 on page 89 during the following discussion. The 256 bytes of PCI-compatible configuration space was so named because it was originally designed for PCI. The first 16 dwords (64 bytes) of this space are the configuration header (Header Type 0 or Header Type 1). Type 0 headers are required for every Function except for the bridge functions that use a Type 1 header. The remaining 48 dwords are used for optional registers including PCI capability structures. For PCIe Functions, some capability structures are required. For example, PCIe Functions must implement the following Capability Structures:

- PCI Express Capability
- Power Management
- MSI and/or MSI-X

4 Address Space & Transaction Routing

The Previous Chapter

The previous chapter provides an introduction to configuration in the PCI Express environment. This includes the space in which a Function's configuration registers are implemented, how a Function is discovered, how configuration transactions are generated and routed, the difference between PCI-compatible configuration space and PCIe extended configuration space, and how software differentiates between an Endpoint and a Bridge.

This Chapter

This chapter describes the purpose and methods of a function requesting address space (either memory address space or IO address space) through Base Address Registers (BARs) and how software must setup the Base/Limit registers in all bridges to route TLPs from a source port to the correct destination port. The general concepts of TLP routing in PCI Express are also discussed, including address-based routing, ID-based routing and implicit routing.

The Next Chapter

The next chapter describes Transaction Layer Packet (TLP) content in detail. We describe the use, format, and definition of the TLP packet types and the details of their related fields.

I Need An Address

Almost all devices have internal registers or storage locations that software (and potentially other devices) need to be able to access. These internal locations may control the device's behavior, report the status of the device, or may be a location to hold data for the device to process. Regardless of the purpose of the internal registers/storage, it is important to be able to access them from outside

the device itself. This means these internal locations need to be *addressable*. Software must be able to perform a read or write operation with an address that will access the appropriate internal location within the targeted device. In order to make this work, these internal locations need to be assigned addresses from one of the address spaces supported in the system.

PCI Express supports the exact same three address spaces that were supported in PCI:

- Configuration
- Memory
- IO

Configuration Space

As we saw in Chapter 1, configuration space was introduced with PCI to allow software to control and check the status of devices in a standardized way. PCI Express was designed to be software backwards compatible with PCI, so configuration space is still supported and used for the same reason as it was in PCI. More info about configuration space (purpose of, how to access, size, contents, etc.) can be found in Chapter 3.

Even though configuration space was originally meant to hold standardized structures (PCI-defined headers, capability structures, etc.), it is very common for PCIe devices to have device-specific registers mapped into their config space. In these cases, the device-specific registers mapped into config space are often control, status or pointer registers as opposed to data storage locations.

Memory and IO Address Spaces

General

In the early days of PCs, the internal registers/storage in IO devices were accessed via IO address space (as defined by Intel). However, because of several limitations and undesirable effects related to IO address space, that we will not be going into here, that address space quickly lost favor with software and hardware vendors. This resulted in the internal registers/storage of IO devices being mapped into memory address space (commonly referred to as memory-mapped IO, or MMIO). However, because early software was written to use IO address space to access internal registers/storage on IO devices, it became common practice to map the same set of device-specific registers in memory

address space as well as in IO address space. This allows new software to access the internal locations of a device using memory address space (MMIO), while allowing legacy (old) software to continue to function because it can still access the internal registers of devices using IO address space.

Newer devices that do not rely on legacy software or have legacy compatibility issues typically just map internal registers/storage through memory address space (MMIO), with no IO address space being requested. In fact, the PCI Express specification actually discourages the use of IO address space, indicating that it is only supported for legacy reasons and may be deprecated in a future revision of the spec.

A generic memory and IO map is shown in Figure 4-1 on page 125. The size of the memory map is a function of the range of addresses that the system can use (often dictated by the CPU addressable range). The size of the IO map in PCIe is limited to 32 bits (4GB), although in many computers using Intel-compatible (x86) processors, only the lower 16 bits (64KB) are used. PCIe can support memory addresses up to 64 bits in size.

The mapping example in Figure 4-1 is only showing MMIO and IO space being claimed by Endpoints, but that ability is not exclusive to Endpoints. It is very common for Switches and Root Complexes to also have device-specific registers accessed via MMIO and IO addresses.

Prefetchable vs. Non-prefetchable Memory Space

Figure 4-1 shows two different types of MMIO being claimed by PCIe devices: Prefetchable MMIO (P-MMIO) and Non-Prefetchable MMIO (NP-MMIO). It's important to describe the distinction between prefetchable and non-prefetchable memory space. Prefetchable space has two very well defined attributes:

- Reads do not have side effects
- Write merging is allowed

Defining a region of MMIO as prefetchable allows the data in that region to be speculatively fetched ahead in anticipation that a Requester might need more data in the near future than was actually requested. The reason it's safe to do this minor caching of the data is that reading the data doesn't change any state info at the target device. That is to say there are no side effects from the act of reading the location. For example, if a Requester asks to read 128 bytes from an address, the Completer might prefetch the next 128 bytes as well in an effort to improve performance by having it on hand when it's requested. However, if the Requester never asks for the extra data, the Completer will eventually have to

discard it to free up the buffer space. If the act of reading the data changed the value at that address (or had some other side effect), it would be impossible to recover the discarded data. However, for prefetchable space, the read had no side effects, so it is always possible to go back and get it later since the original data would still be there.

You may be wondering what sort of memory space might have read side effects? One example would be a memory-mapped status register that was designed to automatically clear itself when read to save the programmer the extra step of explicitly clearing the bits after reading the status.

Making this distinction was more important for PCI than it is for PCIe because transactions in that bus protocol did not include a transfer size. That wasn't a problem when the devices exchanging data were on the same bus, because there was a real-time handshake to indicate when the requester was finished and did not need anymore data, therefore knowing the byte count wasn't so important. But when the transfer had to cross a bridge it wasn't as easy because for reads, the bridge would need to guess the byte count when gathering data on the other bus. Guessing wrong on the transfer size would add latency and reduce performance, so having permission to prefetch could be very helpful. That's why the notion of memory space being designated as prefetchable was helpful in PCI. Since PCIe requests do include a transfer size it's less interesting than it was, but it's carried forward for backward compatibility.

Part Two:

Transaction Layer

5 TLP Elements

The Previous Chapter

The previous chapter describes the purpose and methods of a function requesting address space (either memory address space or IO address space) through Base Address Registers (BARs) and how software must setup the Base/Limit registers in all bridges to route TLPs from a source port to the correct destination port. The general concepts of TLP routing in PCI Express are also discussed, including address-based routing, ID-based routing and implicit routing.

This Chapter

Information moves between PCI Express devices in packets. The three major classes of packets are *Transaction Layer Packets* (TLPs), *Data Link Layer Packets* (DLLPs) and *Ordered Sets*. This chapter describes the use, format, and definition of the variety of TLPs and the details of their related fields. DLLPs are described separately in Chapter 9, entitled "DLLP Elements," on page 307.

The Next Chapter

The next chapter discusses the purposes and detailed operation of the Flow Control Protocol. Flow control is designed to ensure that transmitters never send Transaction Layer Packets (TLPs) that a receiver can't accept. This prevents receive buffer over-runs and eliminates the need for PCI-style inefficiencies like disconnects, retries, and wait-states.

Introduction to Packet-Based Protocol

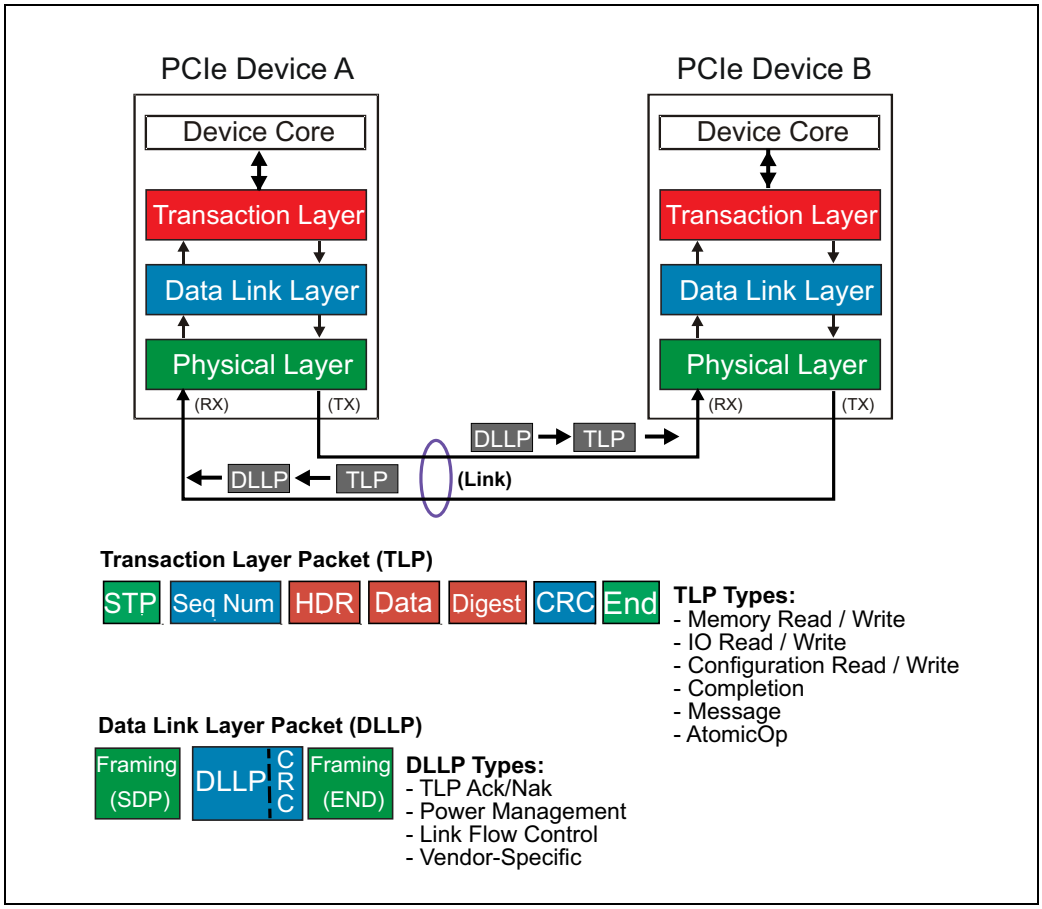
General

Unlike parallel buses, serial transport buses like PCIe use no control signals to identify what's happening on the Link at a given time. Instead, the bit stream they send must have an expected size and a recognizable format to make it pos-

sible for the receiver to understand the content. In addition, PCIe does not use any immediate handshake for the packet while it is being transmitted.

With the exception of the Logical Idle symbols and Physical Layer packets called *Ordered Sets*, information moves across an active PCIe Link in fundamental chunks called packets that are comprised of symbols. The two major classes of packets exchanged are the high-level *Transaction Layer Packets* (TLPs), and low-level Link maintenance packets called *Data Link Layer Packets* (DLLPs). The packets and their flow are illustrated in Figure 5-1 on page 170. Ordered Sets are packets too, however, they are not framed with a start and end symbol like TLPs and DLLPs are. They are also not byte striped like TLPs and DLLPs are. Ordered Set packets are instead replicated on all Lanes of a Link.

Figure 5-1: TLP And DLLP Packets



Motivation for a Packet-Based Protocol

There are three distinct advantages to using a packet-based protocol especially when it comes to data integrity:

1. Packet Formats Are Well Defined

Earlier buses like PCI allow transfers of indeterminate size, making identification of payload boundaries impossible until the end of the transfer. In addition, either device is able to terminate the transfer before it completes, making it difficult for the sender to calculate and send a checksum or CRC covering an entire payload. Instead, PCI uses a simple parity scheme and checks it on each data phase.

By comparison, PCIe packets have a known size and format. The packet *header* at the beginning indicates the packet type and contains the required and optional fields. The size of the header fields is fixed except for the address, which can be 32 bits or 64 bits in size. Once a transfer commences, the recipient can't pause or terminate it early. This structured format allows including information in the TLPs to aid in reliable delivery, including framing symbols, CRC, and a packet Sequence Number.

2. Framing Symbols Define Packet Boundaries

When using 8b/10b encoding in Gen1 and Gen2 mode of operation, each TLP and DLLP packet sent is framed by Start and End control symbols, clearly defining the packet boundaries for the receiver. This is a big improvement over PCI and PCI-X, where the assertion and de-assertion of the single FRAME# signal indicates the beginning and end of a transaction. A glitch on that signal (or any of the other control signals) could cause a target to misconstrue bus events. A PCIe receiver must properly decode a complete 10-bit symbol before concluding Link activity is beginning or ending, so unexpected or unrecognized symbols are more easily recognized and handled as errors.

For the 128b/130b encoding used in Gen3, control characters are no longer employed and there are no framing symbols as such. For more on the differences between Gen3 encoding and the earlier versions, see Chapter 12, entitled "Physical Layer - Logical (Gen3)," on page 407.

3. CRC Protects Entire Packet

Unlike the side-band parity signals used by PCI during the address and data phases of a transaction, the in-band CRC value of PCIe verifies error-free delivery of the entire packet. TLP packets also have a Sequence Number appended to them by the transmitter's Data Link Layer so that if an error is detected at the Receiver, the problem packet can be automatically resent. The transmitter maintains a copy of each TLP sent in a *Retry Buffer* until it has been acknowledged by the receiver. This TLP acknowledgement mechanism, called the *Ack/Nak Protocol*, (and described in Chapter 10, entitled "Ack/Nak Protocol," on page 317) forms the basis of Link-level TLP error detection and correction. This Ack/Nak Protocol error recovery mechanism allows for a timely resolution of the problem at the place or Link where the problem occurred, but requires a local hardware solution to support it.

Transaction Layer Packet (TLP) Details

In PCI Express, high-level transactions originate in the device core of the transmitting device and terminate at the core of the receiving device. The Transaction Layer acts on these requests to assemble outbound TLPs in the Transmitter and interpret them at the Receiver. Along the way, the Data Link Layer and Physical Layer of each device also contribute to the final packet assembly.

TLP Assembly And Disassembly

The general flow of TLP assembly at the transmit side of a Link and disassembly at the receiver is shown in Figure 5-2 on page 173. Let's now walk through the steps from creation of a packet to its delivery to the core logic of the receiver. The key stages in Transaction Layer Packet assembly and disassembly are listed below. The list numbers correspond to the numbers in Figure 5-2 on page 173.

Transmitter:

1. The core logic of Device A sends a request to its PCIe interface. How this is accomplished is outside the scope of the spec or this book. The request includes:
   - Target address or ID (routing information)
   - Source information such as Requester ID and Tag
   - Transaction type/packet type (Command to perform, such as a memory read.)
   - Data payload size (if any) along with data payload (if any)
   - Traffic Class (to assign packet priority)
   - Attributes of the Request (No Snoop, Relaxed Ordering, etc.)

6 Flow Control

The Previous Chapter

The previous chapter discusses the three major classes of packets: *Transaction Layer Packets* (TLPs), *Data Link Layer Packets* (DLLPs) and *Ordered Sets*. This chapter describes the use, format, and definition of the variety of TLPs and the details of their related fields. DLLPs are described separately in Chapter 9, entitled "DLLP Elements," on page 307.

This Chapter

This chapter discusses the purposes and detailed operation of the Flow Control Protocol. Flow control is designed to ensure that transmitters never send Transaction Layer Packets (TLPs) that a receiver can't accept. This prevents receive buffer over-runs and eliminates the need for PCI-style inefficiencies like disconnects, retries, and wait-states.

The Next Chapter

The next chapter discusses the mechanisms that support Quality of Service and describes the means of controlling the timing and bandwidth of different packets traversing the fabric. These mechanisms include application-specific software that assigns a priority value to every packet, and optional hardware that must be built into each device to enable managing transaction priority.

Flow Control Concept

Ports at each end of every PCIe Link must implement Flow Control. Before a packet can be transmitted, flow control checks must verify that the receiving port has sufficient buffer space to accept it. In parallel bus architectures like PCI, transactions are attempted without knowing whether the target is prepared to handle the data. If the request is rejected due to insufficient buffer space, the transaction is repeated (retried) until it completes. This is the "Delayed Transaction Model" of PCI and while it works the efficiency is poor.

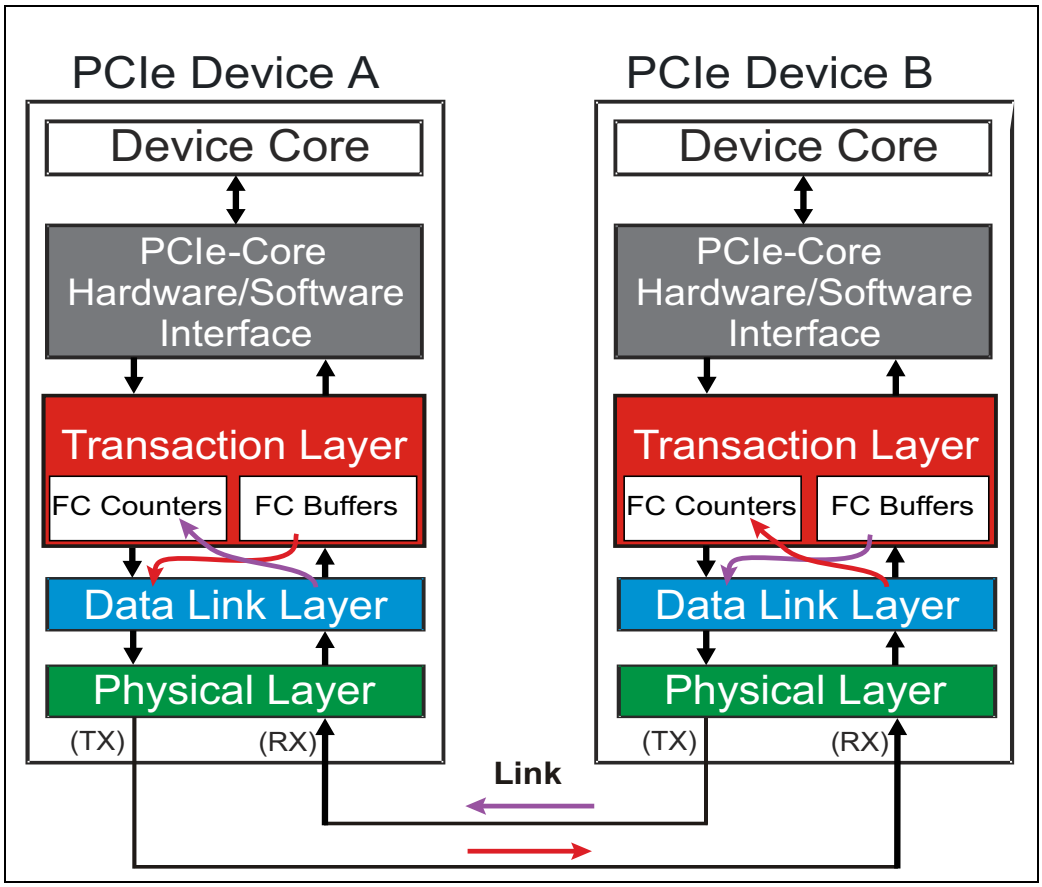
Flow Control mechanisms can improve transmission efficiency if multiple Virtual Channels (VCs) are used. Each Virtual Channel carries transactions that are independent from the traffic flowing in other VCs because flow-control buffers are maintained separately. Therefore, a full Flow Control buffer in one VC will not block access to other VC buffers. PCIe supports up to 8 Virtual Channels.

The Flow Control mechanism uses a credit-based mechanism that allows the transmitting port to be aware of buffer space available at the receiving port. As part of its initialization, each receiver reports the size of its buffers to the transmitter on the other end of the Link, and then during run-time it regularly updates the number of credits available using Flow Control DLLPs. Technically, of course, DLLPs are overhead because they don't convey any data payload, but they are kept small (always 8 symbols in size) to minimize their impact on performance.

Flow control logic is actually a shared responsibility between two layers: the Transaction Layer contains the counters, but the Link Layer sends and receives the DLLPs that convey the information. Figure 6-1 on page 217 illustrates that shared responsibility. In the process of making flow control work:

- **Devices Report Available Buffer Space** — The receiver of each port reports the size of its Flow Control buffers in units called credits. The number of credits within a buffer is sent from the receive-side transaction layer to the transmit-side of the Link Layer. At the appropriate times, the Link Layer creates a Flow Control DLLP to forward this credit information to the receiver at the other end of the Link for each Flow Control Buffer.
- **Receivers Register Credits** — The receiver gets Flow Control DLLPs and transfers the credit values to the transmit-side of the transaction layer. The completes the transfer of credits from one link partner to the other. These actions are performed in both directions until all flow control information has been exchanged.
- **Transmitters Check Credits** — Before it can send a TLP, a transmitter checks the Flow Control Counters to learn whether sufficient credits are available. If so, the TLP is forwarded to the Link Layer but, if not, the transaction is blocked until more Flow Control credits are reported.

Figure 6-1: Location of Flow Control Logic



Flow Control Buffers and Credits

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

VC Flow Control Buffer Organization

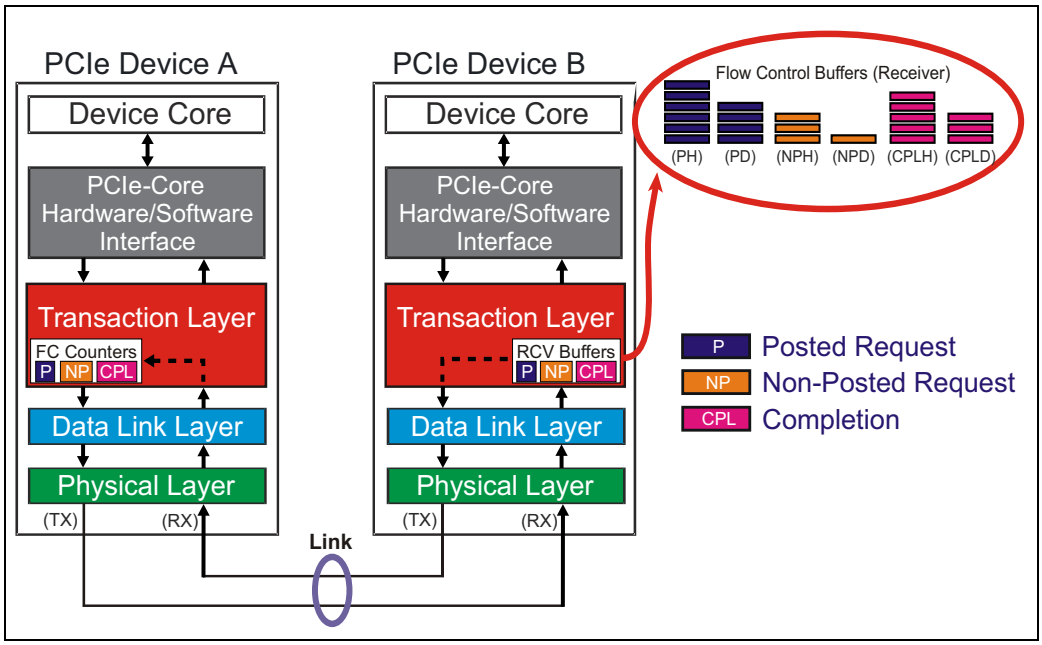
Each VC Flow Control buffer at the receiver is managed for each category of transaction flowing through the virtual channel. These categories are:

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

In addition, each of these categories is separated into header and data portions for transactions that have both header and data. This yields six different buffers each of which implements its own flow control (see Figure 6-2 on page 218).

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

Figure 6-2: Flow Control Buffer Organization



7 Quality of Service

The Previous Chapter

The previous chapter discusses the purposes and detailed operation of the Flow Control Protocol. Flow control is designed to ensure that transmitters never send Transaction Layer Packets (TLPs) that a receiver can't accept. This prevents receive buffer over-runs and eliminates the need for PCI-style inefficiencies like disconnects, retries, and wait-states.

This Chapter

This chapter discusses the mechanisms that support Quality of Service and describes the means of controlling the timing and bandwidth of different packets traversing the fabric. These mechanisms include application-specific software that assigns a priority value to every packet, and optional hardware that must be built into each device to enable managing transaction priority.

The Next Chapter

The next chapter discusses the ordering requirements for transactions in a PCI Express topology. These rules are inherited from PCI. The Producer/Consumer programming model motivated many of them, so its mechanism is described here. The original rules also took into consideration possible deadlock conditions that must be avoided.

Motivation

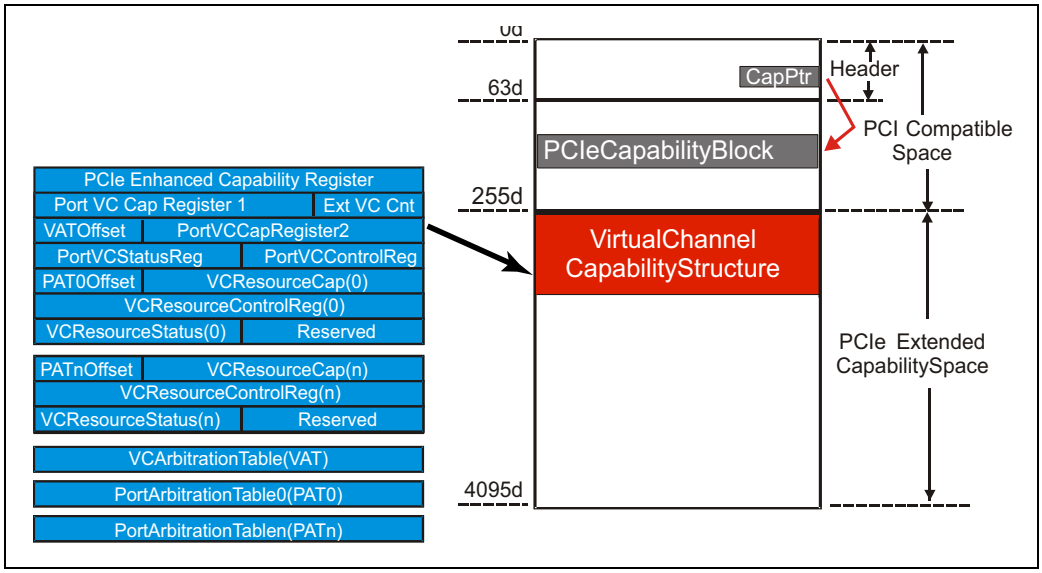
Many computer systems today don't include mechanisms to manage bandwidth for peripheral traffic, but there are some applications that need it. One example is streaming video across a general-purpose data bus, that requires data be delivered at the right time. In embedded guidance control systems timely delivery of video data is also critical to system operation. Foreseeing those needs, the original PCIe spec included Quality of Service (QoS) mechanisms that can give preference to some traffic flows. The broader term for this is

Differentiated Service, since packets are treated differently based on an assigned priority and it allows for a wide range of service preferences. At the high end of that range, QoS can provide predictable and guaranteed performance for applications that need it. That level of support is called "isochronous" service, a term derived from the two Greek words "isos" (equal) and "chronos" (time) that together mean something that occurs at equal time intervals. To make that work in PCIe requires both hardware and software elements.

Basic Elements

Supporting high levels of service places requirements on system performance. For example, the transmission rate must be high enough to deliver sufficient data within a time frame that meets the demands of the application while accommodating competition from other traffic flows. In addition, the latency must be low enough to ensure timely arrival of packets and avoid delay problems. Finally, error handling must be managed so that it doesn't interfere with timely packet delivery. Achieving these goals requires some specific hardware elements, one of which is a set of configuration registers called the Virtual Channel Capability Block as shown in Figure 7-1.

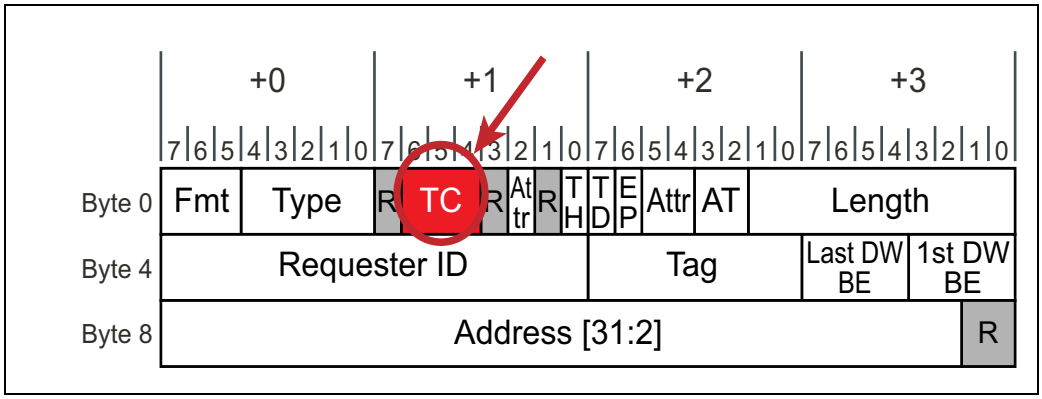
Figure 7-1: Virtual Channel Capability Registers



Traffic Class (TC)

The first thing we need is a way to differentiate traffic; something to distinguish which packets have high priority. This is accomplished by designating Traffic Classes (TCs) that define eight priorities specified by a 3-bit TC field within each TLP header (with ascending priority; TC 0-7). The 32-bit memory request header in Figure 7-2 reveals the location of the TC field. During initialization, the device driver communicates the level of services to the isochronous management software, which returns the appropriate TC values to use for each type of packet. The driver then assigns the correct TC priority for the packet. The TC value defaults to zero so packets that don't need priority service won't accidentally interfere with those that do.

Figure 7-2: Traffic Class Field in TLP Header



Configuration software that's unaware of PCIe won't recognize the new registers and will use the default TC0/VC0 combination for all transactions. In addition, there are some packets that are always required to use TC0/VC0, including Configuration, I/O, and Message transactions. If these packets are thought of as maintenance-level traffic, then it makes sense that they would need to be confined to VC0 and kept out of the path of high-priority packets.

Virtual Channels (VCs)

VCs are hardware buffers that act as queues for outgoing packets. Each port must include the default VC0, but may have as many as eight (from VC0 to VC7). Each channel represents a different path available for outgoing packets.

The motivation for multiple paths is analogous to that of a toll road in which drivers purchase a radio tag that lets them take one of several high priority lanes at the toll booth. Those who don't purchase a tag can still use the road but they'll have to stop at the booth and pay cash each time they go through, and that takes longer. If there was only one path, everyone's access time would be limited by the slowest driver, but having multiple paths available means that those who have priority are not delayed by those who don't.

Assigning TCs to each VC — TC/VC Mapping

The Traffic Class value assigned to each packet travels unchanged to the destination and must be mapped to a VC at each service point as it traverses the path to the target. VC mapping is specific to a Link and can change from one Link to another. Configuration software establishes this association during initialization using the *TC/VC Map* field of the VC Resource Control Register. This 8-bit field permits TC values to be mapped to a selected VC, where each bit position represents the corresponding TC value (bit 0 = TC0, bit 1 = TC1, etc.). Setting a bit assigns the corresponding TC value to the VC ID. Figure 7-3 on page 249 shows a mapping example where TC0 and TC1 are mapped to VC0 and TC2:TC4 are mapped to VC3.

Software has a great deal of flexibility in assigning VC IDs and mapping the TCs, but there are some rules regarding the TC/VC mapping:

- TC/VC mapping must be identical for the two ports attached on either end of the same Link.
- TC0 will automatically be mapped to VC0.
- Other TCs may be mapped to any VC.
- A TC may **not** be mapped to more than one VC.

The number of virtual channels used depends on the greatest capability shared by the two devices attached to a given link. Software assigns an ID for each VC and maps one or more TCs to the VCs.

8 Transaction Ordering

The Previous Chapter

The previous chapter discusses the mechanisms that support Quality of Service and describes the means of controlling the timing and bandwidth of different packets traversing the fabric. These mechanisms include application-specific software that assigns a priority value to every packet, and optional hardware that must be built into each device to enable managing transaction priority.

This Chapter

This chapter discusses the ordering requirements for transactions in a PCI Express topology. These rules are inherited from PCI. The Producer/Consumer programming model motivated many of them, so its mechanism is described here. The original rules also took into consideration possible deadlock conditions that must be avoided.

The Next Chapter

The next chapter describes, Data Link Layer Packets (DLLPs). We describe the use, format, and definition of the DLLP packet types and the details of their related fields. DLLPs are used to support Ack/Nak protocol, power management, flow control mechanism and can be used for vender defined purposes.

Introduction

As with other protocols, PCI Express imposes ordering rules on transactions of the same traffic class (TC) moving through the fabric at the same time. Transactions with different TCs do not have ordering relationships. The reasons for these ordering rules related to transactions of the same TC include:

- Maintaining compatibility with legacy buses (PCI, PCI-X, and AGP).
- Ensuring that the completion of transactions is deterministic and in the sequence intended by the programmer.

- Avoiding deadlock conditions.
- Maximize performance and throughput by minimizing read latencies and managing read and write ordering.

Implementation of the specific PCI/PCIe transaction ordering is based on the following features:

1. Producer/Consumer programming model on which the fundamental ordering rules are based.
2. Relaxed Ordering option that allows an exception to this when the Requester knows that a transaction does not have any dependencies on previous transactions.
3. ID Ordering option that allows a switches to permit requests from one device to move ahead of requests from another device because unrelated threads of execution are being performed by these two devices.
4. Means for avoiding deadlock conditions and supporting PCI legacy implementations.

Definitions

There are three general models for ordering transactions in a traffic flow:

1. **Strong Ordering**: PCI Express requires strong ordering of transactions flowing through the fabric that have the same Traffic Class (TC) assignment. Transactions that have the same TC value assigned to them are mapped to a given VC, therefore the same rules apply to transactions within each VC. Consequently, when multiple TCs are assigned to the same VC all transactions are typically handled as a single TC, even though no ordering relationship exists between different TCs.
2. **Weak Ordering**: Transactions stay in sequence unless reordering would be helpful. Maintaining the strong ordering relationship between transactions can result in all transactions being blocked due to dependencies associated with a given transaction model (e.g., The Producer/Consumer Model). Some of the blocked transactions very likely are not related to the dependencies and can safely be reordered ahead of blocking transactions.
3. **Relaxed Ordering**: Transactions can be reordered, but only under certain controlled conditions. The benefit is improved performance like the weak-ordered model, but only when specified by software so as to avoid problems with dependencies. The drawback is that only some transactions will be optimized for performance. There is some overhead for software to enable transactions for Relaxed Ordering (RO).

Simplified Ordering Rules

The 2.1 revision of the spec introduced a simplified version of the Ordering Table as shown in Table 8-1 on page 289. The table can be segmented on a per topic basis as follows:

- Producer/Consumer rules (page 290)
- Relaxed Ordering rules (page 296)
- Weak Ordering rules (page 299)
- ID Ordering rules (page 301)
- Deadlock avoidance (page 303)

These sections provide details associated with the ordering models, operation, rationales, conditions and requirement.

Ordering Rules and Traffic Classes (TCs)

PCI Express ordering rules apply to transactions of the same Traffic Class (TC). Transactions moving through the fabric that have different TCs have no ordering requirement and are considered to be associated with unrelated applications. As a result, there is no transaction ordering related performance degradation associated with packets of different TCs.

Packets that do share the same TC may experience performance degradation as they flow through the PCIe fabric. This is because switches and devices must support ordering rules that may require packets to be delayed or forwarded in front of packets previously sent.

As discussed in Chapter 7, entitled "Quality of Service," on page 245, transactions of different TC may map to the same VC. The TC-to-VC mapping configuration determines which packets of a given TC map to a specific VC. Even though the transaction ordering rules apply only to packets of the same TC, it may be simpler to design endpoint devices/switches/root complexes that apply the transaction ordering rules to all packets within a VC even though multiple TCs are mapped to the same VC.

As one would expect, there are no ordering relationships between packets that map to different VCs no matter their TC.

Ordering Rules Based On Packet Type

Ordering relationships defined by the PCIe spec are based on TLP type. TLPs are divided into three categories: 1) Posted, 2) Completion and 3) Non-Posted TLPs.

The Posted category of TLPs include memory write requests (MWr) and Messages (Msg/MsgD). Completion category of TLPs include Cpl and CplD. Non-Posted category of TLPs include MRd, IORd, IOWr, CfgRd0, CfgRd1, CfgWr0 and CfgWr1.

The transaction ordering rules are described by a table in the following section "The Simplified Ordering Rules Table" on page 288. As you will notice, the table shows TLPs listed according to the three categories mentioned above with their ordering relationships defined.

The Simplified Ordering Rules Table

The table is organized in a Row Pass Column fashion. All of the rules are summarized following the Simplified Ordering Table. Each rule or group of rules define the actions that are required.

In Table 8-1 on page 289, columns 2 - 5 represent transactions that have previously been delivered by a PCI Express device, while row A - D represents a new transaction that has just arrived. For outbound transactions, the table specifies whether a transaction represented in the row (A - D) is allowed to pass a previous transaction represented by the column (2 - 5). A 'No' entry means the transaction in the row is not allowed to pass the transaction in the column. A 'Yes' entry means the transaction in the row must be allowed to pass the transaction in the column to avoid a deadlock. A 'Yes/No' entry means a transaction in a row is allowed to pass the transaction in the column but is not required to do so. The entries in the following have the meaning.

Part Three:

Data Link Layer

9 DLLP Elements

The Previous Chapter

The previous chapter discussed the ordering requirements for transactions in a PCI Express topology. These rules are inherited from PCI, and the Producer/Consumer programming model motivated many of them, so its mechanism is described here. The original rules also took into consideration possible deadlock conditions that must be avoided, but did not include any means to avoid the performance problems that could result.

This Chapter

In this chapter we describe the other major category of packets, *Data Link Layer Packets* (DLLPs). We describe the use, format, and definition of the DLLP packet types and the details of their related fields. DLLPs are used to support Ack/Nak protocol, power management, flow control mechanism and can even be used for vendor-defined purposes.

The Next Chapter

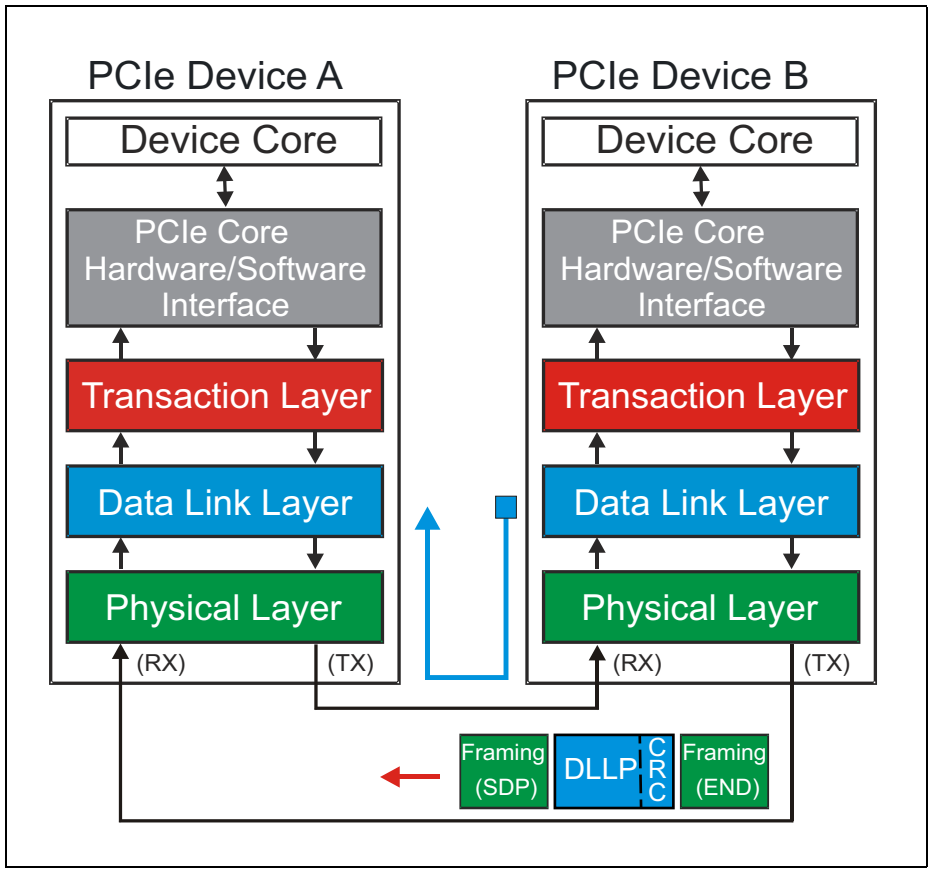
The following chapter describes a key feature of the Data Link Layer: an automatic, hardware-based mechanism for ensuring reliable transport of TLPs across the Link. Ack DLLPs confirm good reception of TLPs while Nak DLLPs indicate a transmission error. We describe the normal rules of operation when no TLP or DLLP error is detected as well as error recovery mechanisms associated with both TLP and DLLP errors.

General

The Data Link Layer can be thought of as managing the lower level Link protocol. Its primary responsibility is to assure the integrity of TLPs moving between devices, but it also plays a part in TLP flow control, Link initialization and power management, and conveys information between the Transaction Layer above it and the Physical Layer below it.

In performing these jobs, the Data Link Layer exchanges packets with its neighbor known as Data Link Layer Packets (DLLPs). DLLPs are communicated between the Data Link Layers of each device. Figure 9-1 on page 308 illustrates a DLLP exchanged between devices.

Figure 9-1: Data Link Layer Sends A DLLP



DLLPs Are Local Traffic

DLLPs have a simple packet format and are a fixed size, 8 bytes total, including the framing bytes. Unlike TLPs, they carry no target or routing information because they are only used for nearest-neighbor communications and don't get routed at all. They're also not seen by the Transaction Layer since they're not part of the information exchanged at that level.

Receiver handling of DLLPs

When DLLPs are received, several rules apply:

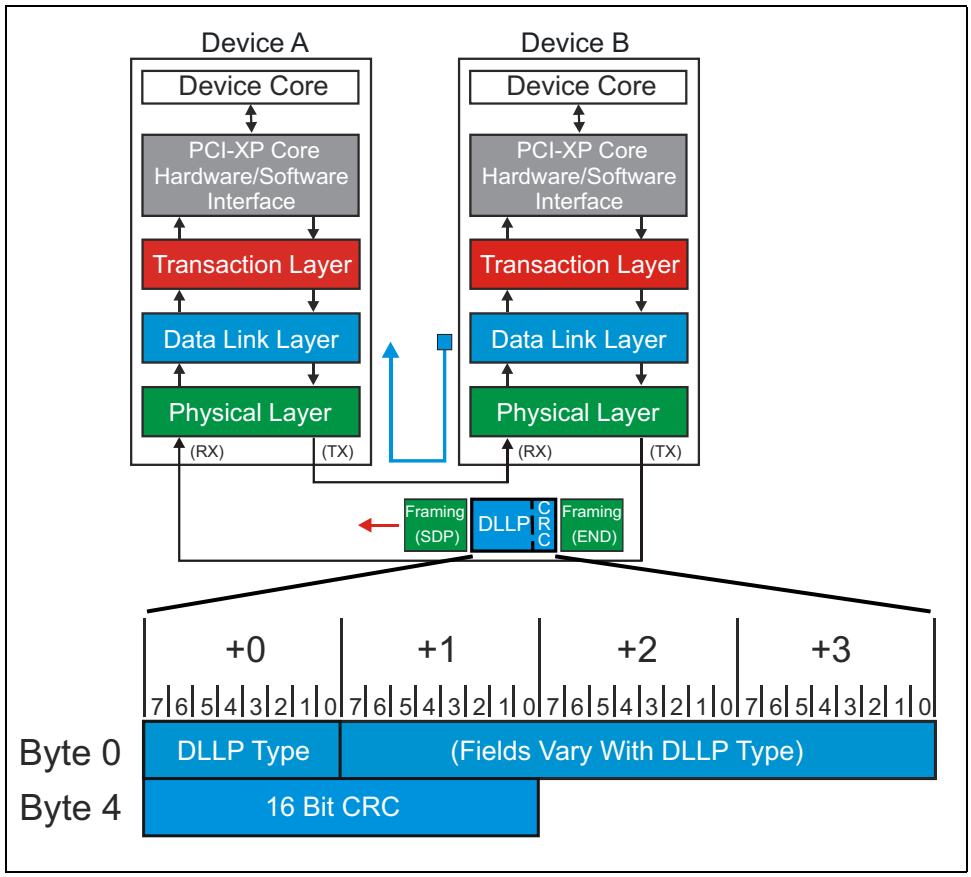
1. They're immediately processed at the Receiver. In other words, their flow cannot be controlled the way it is for TLPs (DLLPs are not subject to flow control).
2. They're checked for errors; first at the Physical Layer, and then at the Data Link Layer. The 16-bit CRC included with the packet is checked by calculating what the CRC should be and comparing it to the received value. DLLPs that fail this check are discarded. How will the Link recover from this error? DLLPs still arrive periodically, and the next one of that type that succeeds will update the missing information.
3. Unlike TLPs, there's no acknowledgement protocol for DLLPs. Instead, the spec defines time-out mechanisms to facilitate recovery from failed DLLPs.
4. If there are no errors, the DLLP type is determined and passed to the appropriate internal logic to manage:
   - Ack/Nak notification of TLP status
   - Flow Control notification of buffer space available
   - Power Management settings
   - Vendor specific information

Sending DLLPs

General

These packets originate at the Data Link Layer and are passed to the Physical Layer. If 8b/10b encoding is in use (Gen1 and Gen2 mode), framing symbols will be added to both ends of the DLLP at this level before the packet is sent. In Gen3 mode, a SDP token of two bytes is added to the front end of the DLLP, but no END is added to the end of the DLLP. Figure 9-2 on page 310 shows a generic (Gen1/Gen2) DLLP in transit, showing the framing symbols and the general contents of the packet.

Figure 9-2: Generic Data Link Layer Packet Format



DLLP Packet Size is Fixed at 8 Bytes

Data Link Layer Packets are always 8 bytes long for both 8b/10b and 128b/130b and consist of the following components:

1. A 1 DW core (4 bytes) containing the one-byte DLLP Type field and three additional bytes of attributes. The attributes vary with the DLLP type.
2. A 2-byte CRC value that is calculated based on the core contents of the DLLP. It is important to point out that this CRC is different from the LCRCs added to TLPs. This CRC is only 16 bits in size and is calculated differently than the 32-bit LCRCs in TLPs. This CRC is appended to the core DLLP and then these 6 bytes are passed to the Physical Layer.

10 Ack/Nak Protocol

The Previous Chapter

In the previous chapter we describe *Data Link Layer Packets* (DLLPs). We describe the use, format, and definition of the DLLP types and the details of their related fields. DLLPs are used to support Ack/Nak protocol, power management, flow control mechanism and can be used for vendor-defined purposes.

This Chapter

This chapter describes a key feature of the Data Link Layer: an automatic, hardware-based mechanism for ensuring reliable transport of TLPs across the Link. Ack DLLPs confirm successful reception of TLPs while Nak DLLPs indicate a transmission error. We describe the normal rules of operation when no TLP or DLLP error is detected as well as error recovery mechanisms associated with both TLP and DLLP errors.

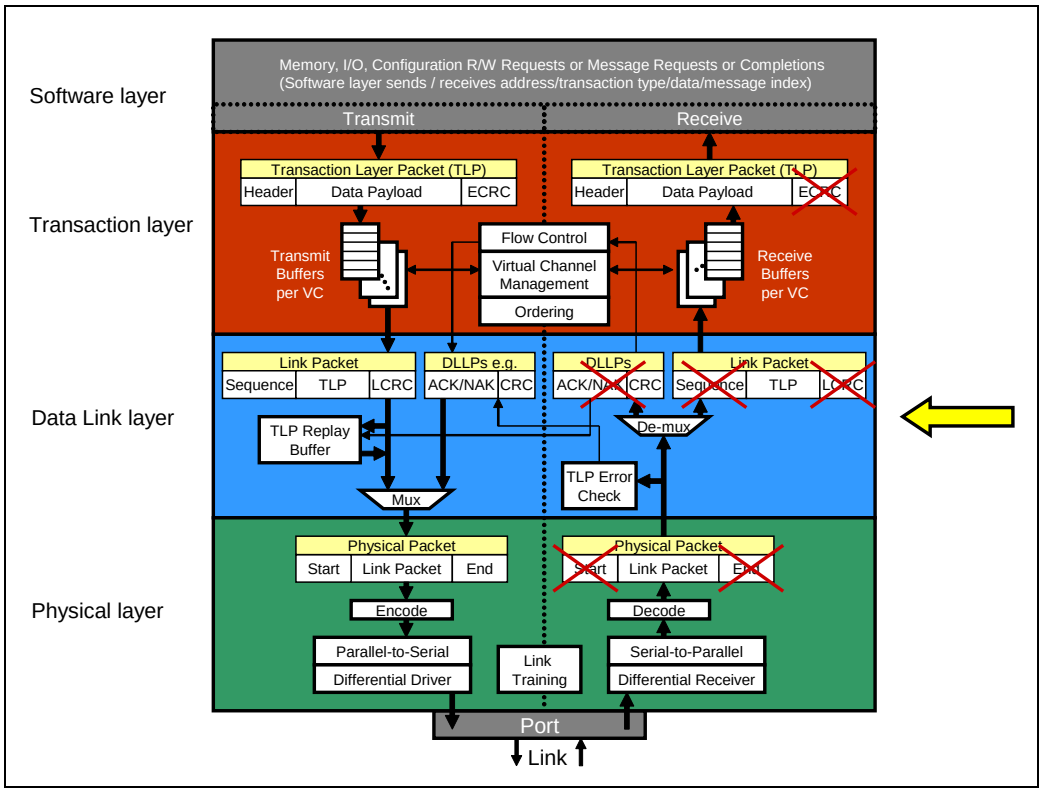
The Next Chapter

The next chapter describes the Logical sub-block of the Physical Layer, which prepares packets for serial transmission and reception. Several steps are needed to accomplish this and they are described in detail. This chapter covers the logic associated with the first two spec versions Gen1 and Gen2 that use 8b/10b encoding. The logic for Gen3 does not use 8b/10b encoding and is described separately in the chapter called "Physical Layer - Logical (Gen3)" on page 407.

Goal: Reliable TLP Transport

The function of the Data Link Layer (shown in Figure 10-1 on page 318) is to ensure reliable delivery of TLPs. The spec requires a BER (Bit Error Rate) of no worse than 10^{-12}, but errors will still happen often enough to cause trouble, and a single bit error will corrupt an entire packet. This problem will only become more pronounced as Link rates continue to increase with new generations.

Figure 10-1: Data Link Layer

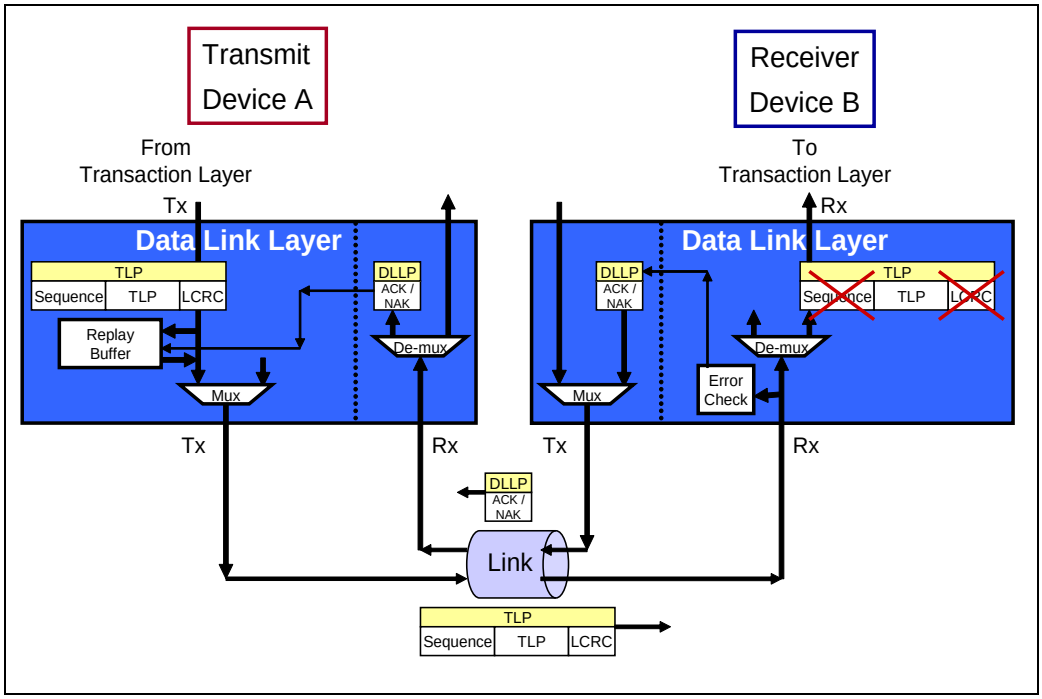


To facilitate this goal, an error detection code called an LCRC (Link Cyclic Redundancy Code) is added to each TLP. The first step in error checking is simply to verify that this code still evaluates correctly at the receiver. If each packet is given a unique incremental Sequence Number as well, then it will be easy to sort out which packet, out of several that have been sent, encountered an error. Using that Sequence Number, we can also require that TLPs must be successfully received in the same order they were sent. This simple rule makes it easy to detect missing TLPs at the Receiver's Data Link Layer.

The basic blocks in the Data Link Layer associated with the Ack/Nak protocol are shown in greater detail in Figure 10-2 on page 319. Every TLP sent across the Link is checked at the receiver by evaluating the LCRC (first) and Sequence Number (second) in the packet. The receiving device notifies the transmitting device that a good TLP has been received by returning an Ack. Reception of an

Ack at the transmitter means that the receiver has received at least one TLP successfully. On the other hand, reception of a Nak by the transmitter indicates that the receiver has received at least one TLP in error. In that case, the transmitter will re-send the appropriate TLP(s) in hopes of a better result this time. This is sensible, because things that would cause a transmission error would likely be transient events and a replay will have a very good chance of solving the problem.

Figure 10-2: Overview of the Ack/Nak Protocol



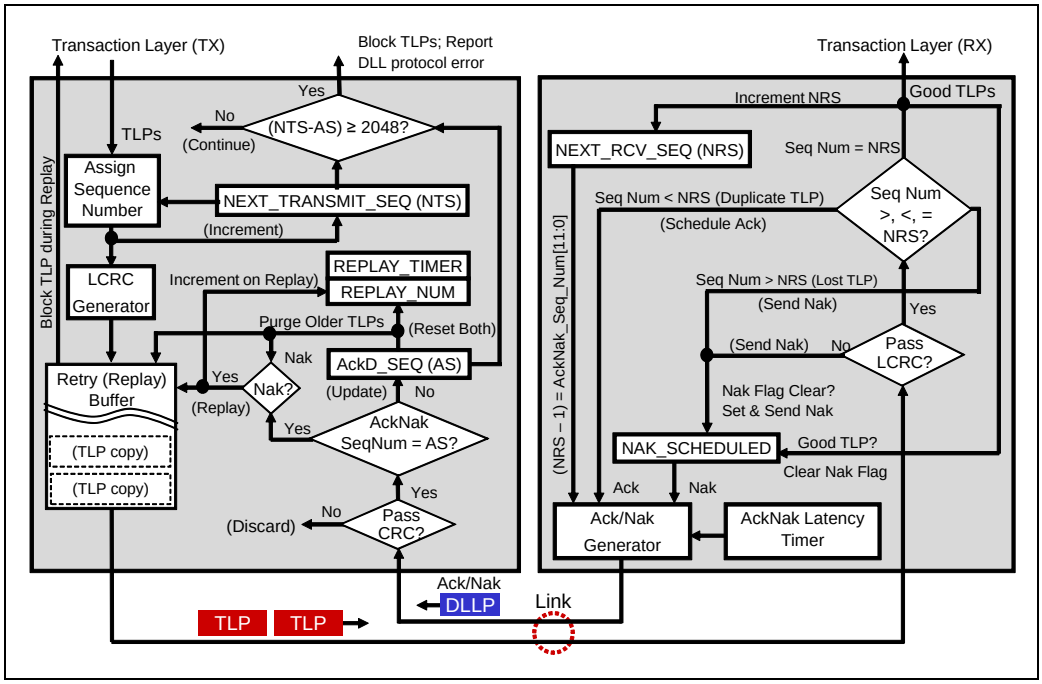
Since both the sending and receiving devices in the protocol have both a transmit and a receive side, this chapter will use the terms:

- **Transmitter** to mean the device that sends TLPs
- **Receiver** to mean the device that receives TLPs

Elements of the Ack/Nak Protocol

The major Ack/Nak protocol elements of the Data Link Layer are shown in Figure 10-3 on page 320. There's too much to consider all at once, though, so let's begin by focusing on just the transmitter elements, which are shown in a larger view in Figure 10-4 on page 322.

Figure 10-3: Elements of the Ack/Nak Protocol



Transmitter Elements

As TLPs arrive from the Transaction Layer, several things are done to prepare them for robust error detection at the receiver. As shown in the diagram TLPs are first assigned the next sequential Sequence Number, obtained from the 12-bit NEXT_TRANSMIT_SEQ counter.

11 Physical Layer - Logical (Gen1 and Gen2)

The Previous Chapter

The previous chapter describes the Ack/Nak Protocol: an automatic, hardware-based mechanism for ensuring reliable transport of TLPs across the Link. Ack DLLPs confirm good reception of TLPs while Nak DLLPs indicate a transmission error. The chapter describes the normal rules of operation as well as error recovery mechanisms.

This Chapter

This chapter describes the Logical sub-block of the Physical Layer. This prepares packets for serial transmission and recovery. Several steps are needed to accomplish this and they are described in detail. This chapter covers the logic associated with the Gen1 and Gen2 protocol that use 8b/10b encoding. The logic for Gen3 does not use 8b/10b encoding and is described separately in the chapter called "Physical Layer - Logical (Gen3)" on page 407.

The Next Chapter

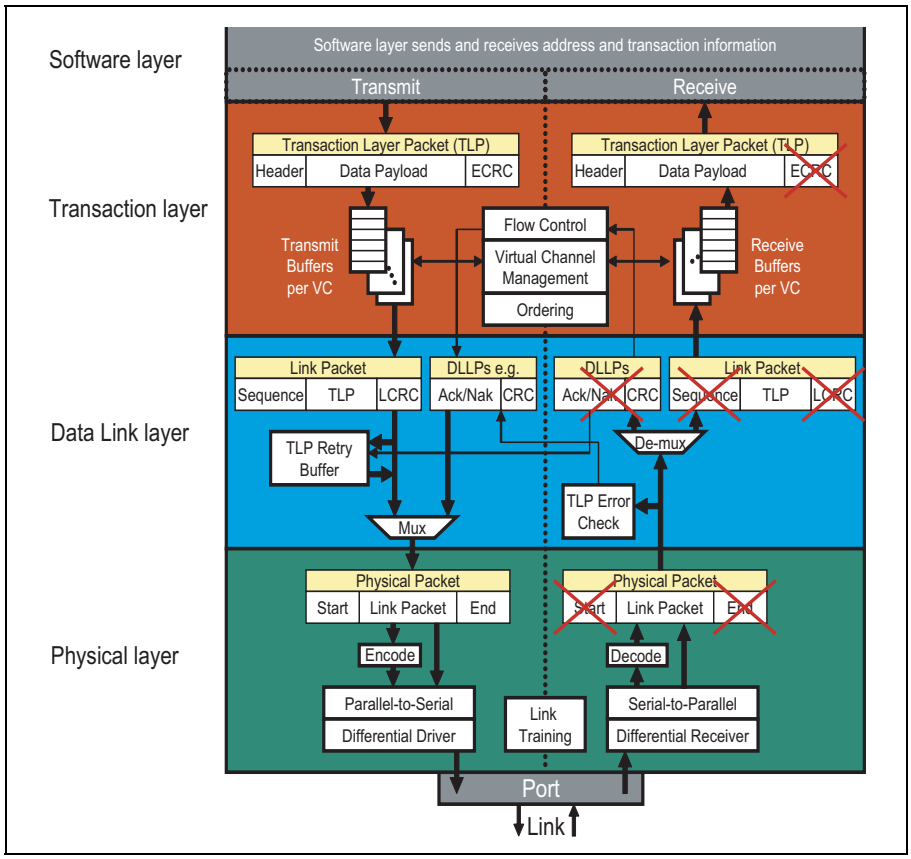
The next chapter describes the Physical Layer characteristics for the third generation (Gen3) of PCIe. The major change includes the ability to double the bandwidth relative to Gen2 without needing to double the frequency by eliminating the need for 8b/10b encoding. More robust signal compensation is necessary at Gen3 speed. Making these changes is more complex than might be expected.

Physical Layer Overview

This Physical Layer Overview introduces the relationships between the Gen1, Gen2 and Gen3 implementations. Thereafter the focus is the logical Physical Layer implementation associated with Gen1 and Gen2. The logical Physical Layer implementation for Gen3 is described in the next chapter.

The Physical Layer resides at the bottom of the interface between the external physical link and Data Link Layer. It converts outbound packets from the Data Link Layer into a serialized bit stream that is clocked onto all Lanes of the Link. This layer also recovers the bit stream from all Lanes of the Link at the receiver. The receive logic de-serializes the bits back into a Symbol stream, re-assembles the packets, and forwards TLPs and DLLPs up to the Data Link Layer.

Figure 11-1: PCIe Port Layers



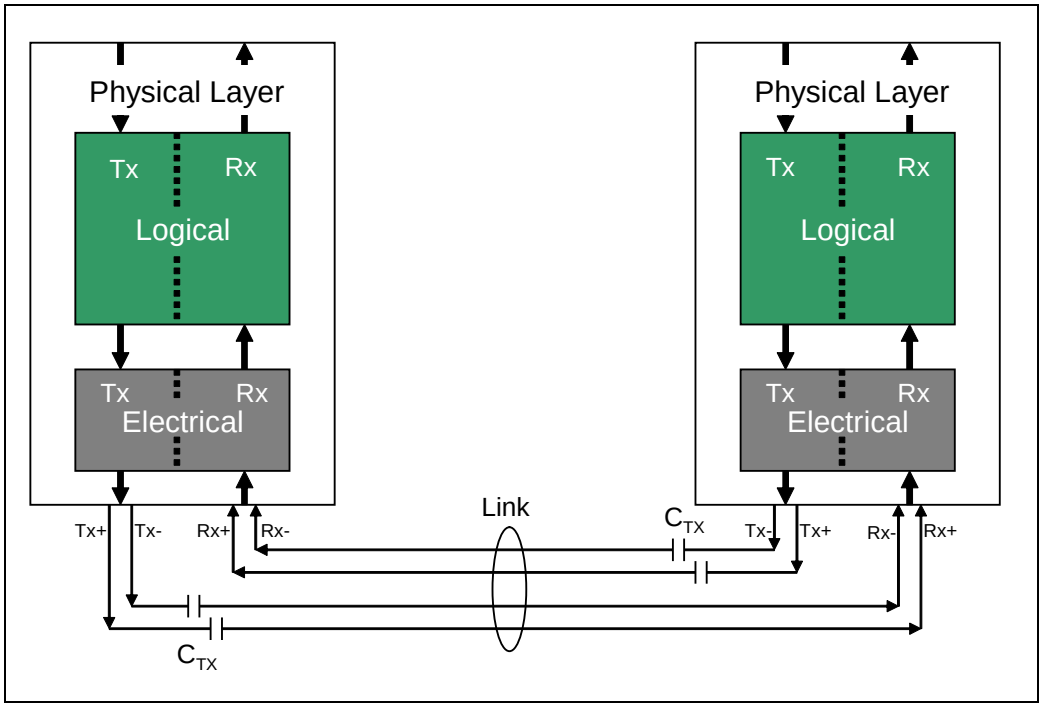
The contents of the layers are conceptual and don't define precise logic blocks, but to the extent that designers do partition them to match the spec their implementations can benefit because of the constantly increasing data rates affect the Physical Layer more than the others. Partitioning a design by layered responsibilities allows the Physical Layer to be adapted to the higher clock rates while changing as little as possible in the other layers.

The 3.0 revision of the PCIe spec does not use specific terms to distinguish the different transmission rates defined by the versions of the spec. With that in mind, the following terms are defined and used in this book.

- **Gen1** - the first generation of PCIe (rev 1.x) operating at 2.5 GT/s
- **Gen2** - the second generation (rev 2.x) operating at 5.0 GT/s
- **Gen3** - the third generation (rev 3.x) operating at 8.0 GT/s

The Physical Layer is made up of two sub-blocks: the Logical part and the Electrical part as shown in Figure 11-2. Both contain independent transmit and receive logic, allowing dual-simplex communication.

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



Observation

The spec describes the functionality of the Physical Layer but is purposefully vague regarding implementation details. Evidently, the spec writers were reluctant to give details or example implementations because they wanted to leave room for individual vendors to add value with clever or creative versions of the logic. For our discussion though, an example is indispensable, and one was chosen that illustrates the concepts. It's important to make clear that this example has not been tested or validated, nor should a designer feel compelled to implement a Physical Layer in such a manner.

Transmit Logic Overview

For simplicity, let's begin with a high-level overview of the transmit side of this layer, shown in Figure 11-3 on page 365. Starting at the top, we can see that packet bytes entering from the Data Link layer first go into a buffer. It makes sense to have a buffer here because there will be times when the packet flow from the Data Link Layer must be delayed to allow Ordered Set packets and other items to be injected into the flow of bytes.

For Gen1 and Gen2 operation, these injected items are control and data characters used to mark packet boundaries and create ordered sets. To differentiate between these two types of characters, a D/K# bit (Data or "Kontrol") is added. The logic can see what value D/K# should take on based on the source of the character.

Gen3 mode of operation, doesn't use control characters, so data patterns are used to make up the ordered sets that identify if transmitted bytes are associated with TLPs / DLLPs or Ordered Sets. A 2-bit Sync Header is inserted at the beginning of a 128 bit (16 byte) block of data. The Sync Header informs the receiver whether the received block is a Data Block (TLP or DLLP related bytes) or an Ordered Set Block. Since there are no control characters in Gen3 mode, the D/K# bit is not needed.

12 Physical Layer - Logical (Gen3)

The Previous Chapter

The previous chapter describes the Gen1/Gen2 logical sub-block of the Physical Layer. This layer prepares packets for serial transmission and recovery, and the several steps needed to accomplish this are described in detail. The chapter covers logic associated with the Gen1 and Gen2 protocol that use 8b/10b encoding/decoding.

This Chapter

This chapter describes the logical Physical Layer characteristics for the third generation (Gen3) of PCIe. The major change includes the ability to double the bandwidth relative to Gen2 speed without needing to double the frequency (Link speed goes from 5 GT/s to 8 GT/s). This is accomplished by eliminating 8b/10b encoding when in Gen3 mode. More robust signal compensation is necessary at Gen3 speed.

The Next Chapter

The next chapter describes the Physical Layer electrical interface to the Link. The need for signal equalization and the methods used to accomplish it are also discussed here. This chapter combines electrical transmitter and receiver characteristics for both Gen1, Gen2 and Gen3 speeds.

Introduction to Gen3

Recall that when a PCIe Link enters training (i.e., after a reset) it always begins using Gen1 speed for backward compatibility. If higher speeds were advertised during the training, the Link will immediately transition to the Recovery state and attempt to change to the highest commonly-supported speed.

The major motivation for upgrading the PCIe spec to Gen3 was to double the bandwidth, as shown in Table 12-1 on page 408. The straightforward way to accomplish this would have been to simply double the signal frequency from 5 GT/s to 10 Gb/s, but doing that presented several problems:

- Higher frequencies consume substantially more power, a condition exacerbated by the need for sophisticated conditioning logic (equalization) to maintain signal integrity at the higher speeds. In fact, the power demand of this equalizing logic is mentioned in PCISIG literature as a big motivation for keeping the frequency as low as practical.
- Some circuit board materials experience significant signal degradation at higher frequencies. This problem can be overcome with better materials and more design effort, but those add cost and development time. Since PCIe is intended to serve a wide variety of systems, the goal was that it should work well in inexpensive designs, too.
- Similarly, allowing new designs to use the existing infrastructure (circuit boards and connectors, for example) minimizes board design effort and cost. Using higher frequencies makes that more difficult because trace lengths and other parameters must be adjusted to account for the new timing, and that makes high frequencies less desirable.

Table 12-1: PCI Express Aggregate Bandwidth for Various Link Widths

Link Width	x1	x2	x4	x8	x12	x16	x32
Gen1 Bandwidth (GB /s)	0.5	1	2	4	6	8	16
Gen2 Bandwidth (GB/s)	1	2	4	8	12	16	32
Gen3 Bandwidth (GB/s)	2	4	8	16	24	32	64

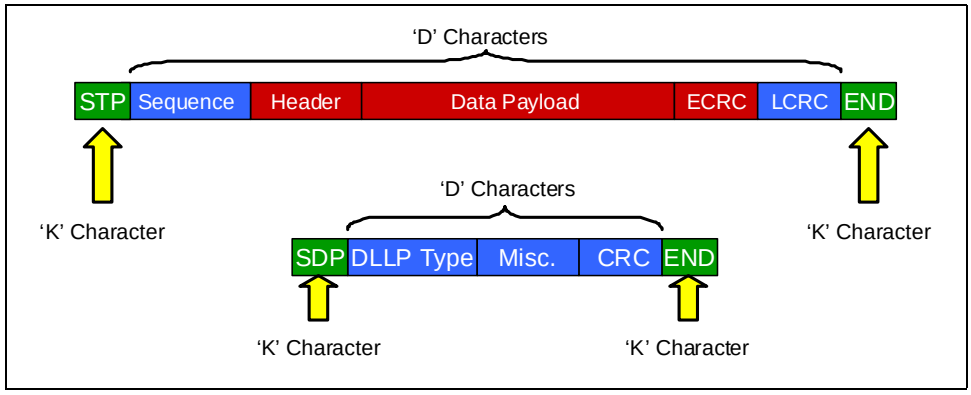
These considerations led to two significant changes to the Gen3 spec compared with the previous generations: a new encoding model and a more sophisticated signal equalization model.

New Encoding Model

The logical part of the Physical Layer replaced the 8b/10b encoding with a new 128b/130b encoding scheme. Of course, this meant departing from the well-understood 8b/10b model used in many serial designs. Designers were willing to take this step to recover the 20% transmission overhead imposed by the 8b/10b encoding. Using 128b/130b means the Lanes are now delivering 8 bits/byte instead of 10 bits, and that means an 8.0 GT/s data rate that doubles the bandwidth. This equates to a bandwidth of 1 GB/s in each direction.

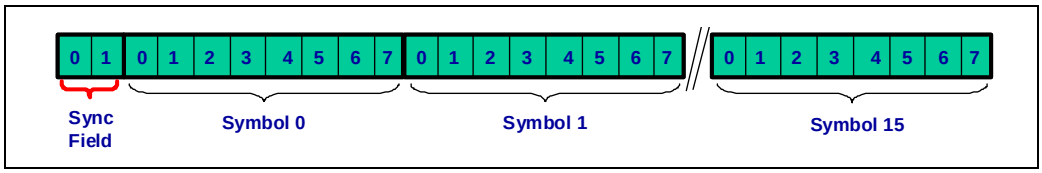
To illustrate the difference between these two encodings, first consider Figure 12-1 that shows the general 8b/10b packet construction. The arrows highlight the Control (K) characters representing the framing Symbols for the 8b/10b packets. Receivers know what to expect by recognizing these control characters. See “8b/10b Encoding” on page 380 to review the benefits of this encoding scheme.

Figure 12-1: 8b/10b Lane Encoding



By comparison, Figure 12-2 on page 410 shows the 128b/130b encoding. This encoding does not affect bytes being transferred, instead the characters are grouped into blocks of 16 bytes with a 2-bit Sync field at the beginning of each block. The 2-bit Sync field specifies whether the block includes Data (10b) or Ordered Sets (01b). Consequently, the Sync field indicates to the receiver what kind of traffic to expect and when it will begin. Ordered sets are similar to the 8b/10b version in that they must be driven on all the Lanes simultaneously. That requires getting the Lanes properly synchronized and this is part of the training process (see “Achieving Block Alignment” on page 438).

Figure 12-2: 128b/130b Block Encoding



Sophisticated Signal Equalization

The second change is made to the electrical sub-block of the Physical Layer and involves more sophisticated signal equalization both at the transmit side of the Link and optionally at the receiver. Gen1 and Gen2 implementations use a fixed Tx de-emphasis to achieve good signal quality. However, increasing transmission frequencies beyond 5 GT/s causes signal integrity problems to become more pronounced, requiring more transmitter and receiver compensation. This can be managed somewhat at the board level but the designers wanted to allow the external infrastructure to remain the same as much as possible, and instead placed the burden on the PHY transmitter and receiver circuits. For more details on signal conditioning, refer to “Solution for 8.0 GT/s - Transmitter Equalization” on page 474.

Encoding for 8.0 GT/s

As previously discussed, the Gen3 128b/130b encoding method uses Link-wide packets and per-Lane block encoding. This section provides additional details regarding the encoding.

Lane-Level Encoding

To illustrate the use of Blocks, consider Figure 12-3 on page 411, where a single-Lane Data Block is shown. At the beginning are the two Sync Header bits follower by 16 bytes (128 bits) of information resulting in 130 transmitted bits. The Sync Header simply defines whether a Data block (10b) or an Ordered Set (01b) is being sent. You may have noticed the Data Block in Figure 12-3 has a Sync Header value of 01 rather than the 10b value mentioned above. This is because the least significant bit of the Sync Header is sent first when transmitting the block across the link. Notice the symbols following the Sync Header are also sent with the least significant bit first.

13 Physical Layer - Electrical

The Previous Chapter

The previous chapter describes the logical Physical Layer characteristics for the third generation (Gen3) of PCIe. The major change includes the ability to double the bandwidth relative to Gen2 speed without needing to double the frequency (Link speed goes from 5 GT/s to 8 GT/s). This is accomplished by eliminating 8b/10b encoding when in Gen3 mode. More robust signal compensation is necessary at Gen3 speed. Making these changes is more complex than might be expected.

This Chapter

This chapter describes the Physical Layer electrical interface to the Link, including some low-level characteristics of the differential Transmitters and Receivers. The need for signal equalization and the methods used to accomplish it are also discussed here. This chapter combines electrical transmitter and receiver characteristics for both Gen1, Gen2 and Gen3 speeds.

The Next Chapter

The next chapter describes the operation of the Link Training and Status State Machine (LTSSM) of the Physical Layer. The initialization process of the Link is described from Power-On or Reset until the Link reaches the fully-operational L0 state during which normal packet traffic occurs. In addition, the Link power management states L0s, L1, L2, L3 are discussed along with the causes of transitions between the states. The Recovery state during which bit lock, symbol lock or block lock can be re-established is described.

Backward Compatibility

The spec begins the Physical Layer Electrical section with the observation that newer data rates need to be backward compatible with the older rates. The following summary defines the requirements:

- Initial training is done at 2.5 GT/s for all devices.
- Changing to other rates requires negotiation between the Link partners to determine the peak common frequency.
- Root ports that support 8.0 GT/s are required to support both 2.5 and 5.0 GT/s as well.
- Downstream devices must obviously support 2.5 GT/s, but all higher rates are optional. This means that an 8 GT/s device is not required to support 5 GT/s.

In addition, the optional Reference clock (Refclk) remains the same regardless of the data rate and does not require improved jitter characteristics to support the higher rates.

In spite of these similarities, the spec does describe some changes for the 8.0 GT/s rate:

- **ESD standards:** Earlier PCIe versions required all signal and power pins to withstand a certain level of ESD (Electro-Static Discharge) and that's true for the 3.0 spec, too. The difference is that more JEDEC standards are listed and the spec notes that they apply to devices regardless of which rates they support.
- **Rx powered-off Resistance:** The new impedance values specified for 8.0 GT/s ($Z_{RX\text{-}HIGH\text{-}IMP\text{-}DC\text{-}POS}$ and $Z_{RX\text{-}HIGH\text{-}IMP\text{-}DC\text{-}NEG}$) will be applied to devices supporting 2.5 and 5.0 GT/s as well.
- **Tx Equalization Tolerance:** Relaxing the previous spec tolerance on the Tx de-emphasis values from +/- 0.5 dB to +/- 1.0 dB makes the -3.5 and -6.0 dB de-emphasis tolerance consistent across all three data rates.
- **Tx Equalization during Tx Margining:** The de-emphasis tolerance was already relaxed to +/- 1.0 dB for this case in the earlier specs. The accuracy for 8.0 GT/s is determined by the Tx coefficient granularity and the TxEQ tolerances for the Transmitter during normal operation.
- **$V_{TX\text{-}ACCM}$ and $V_{RX\text{-}ACCM}$:** For 2.5 and 5.0 GT/s these are relaxed to 150 mVPP for the Transmitter and 300 mVPP for the Receiver.

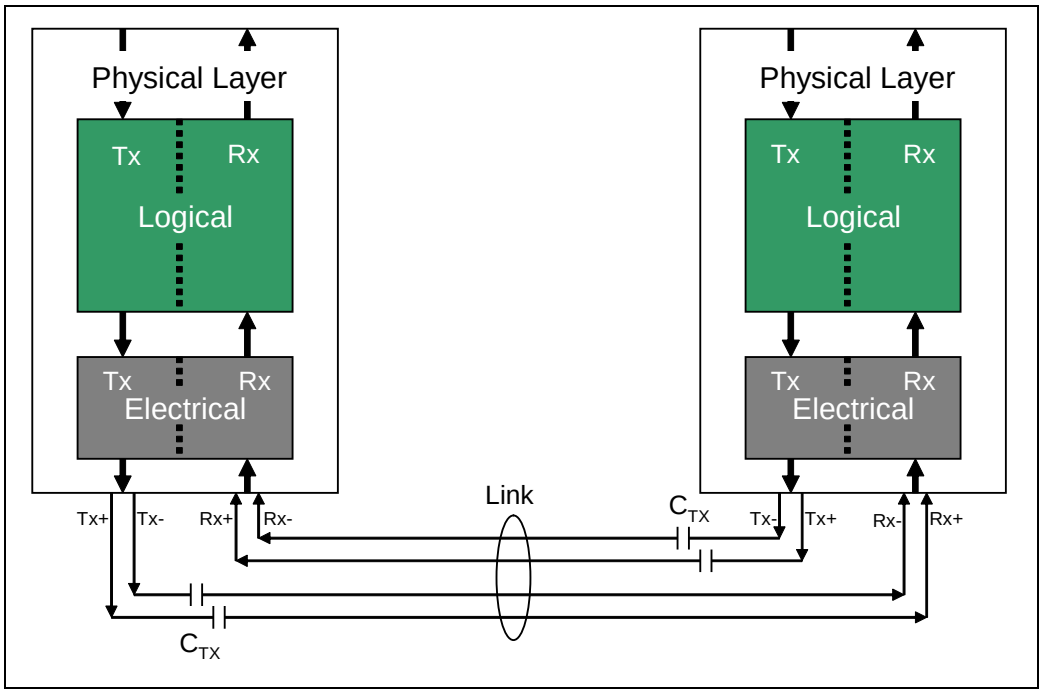
Component Interfaces

Components from different vendors must work reliably together, so a set of parameters are specified that must be met for the interface. For 2.5 GT/s it was implied, and for 5.0 GT/s it was explicitly stated, that the characteristics of this interface are defined at the device pins. That allows a component to be characterized independently, without requiring the use of any other PCIe components. Other interfaces may be specified at a connector or other location, but those are not covered in the base spec and would be described in other form-factor specs like the *PCI Express Card Electromechanical Spec*.

Physical Layer Electrical Overview

The electrical sub-block associated with each lane, as shown in Figure 13-1 on page 450, provides the physical interface to the Link and contains differential Transmitters and Receivers. The Transmitter delivers outbound Symbols on each Lane by converting the bit stream into two single-ended electrical signals with opposite polarity. Receivers compare the two signals and, when the difference is sufficiently positive or negative, generate a one or zero internally to represent the intended serial bit stream to the rest of the Physical Layer.

Figure 13-1: Electrical Sub-Block of the Physical Layer



When the Link is in the L0 full-on state, the drivers apply the differential voltage associated with a logical 1 and logical 0 while maintaining the correct DC common mode voltage. Receivers sense this voltage as the input stream, but if it drops below a threshold value, it's understood to represent the Electrical Idle Link condition. Electrical Idle is entered when the Link is disabled, or when ASPM logic puts the Link into low-power Link states such as L0s or L1 (see "Electrical Idle" on page 736 for more on this topic).

Devices must support the Transmitter equalization methods required for each supported data rate so they can achieve adequate signal integrity. De-emphasis is applied for 2.5 and 5.0 GT/s, and a more complex equalization process is applied for 8.0 GT/s. These are described in more detail in "Signal Compensation" on page 468, and "Recovery.Equalization" on page 587.

The drivers and Receivers are short-circuit tolerant, making PCIe add-in cards suited for hot (powered-on) insertion and removal events in a hot-plug environment. The Link connecting two components is AC-coupled by adding a capacitor in-line, typically near the Transmitter side of the Link. This serves to de-

14 Link Initialization & Training

The Previous Chapter

The previous chapter describes the Physical Layer electrical interface to the Link, including some low-level characteristics of the differential Transmitters and Receivers. The need for signal equalization and the methods used to accomplish it are also discussed here. This chapter combines electrical transmitter and receiver characteristics for both Gen1, Gen2 and Gen3 speeds.

This Chapter

This chapter describes the operation of the Link Training and Status State Machine (LTSSM) of the Physical Layer. The initialization process of the Link is described from Power-On or Reset until the Link reaches fully-operational L0 state during which normal packet traffic occurs. In addition, the Link power management states L0s, L1, L2, and L3 are discussed along with the state transitions. The Recovery state, during which bit lock, symbol lock or block lock are re-established is described. Link speed and width change for Link bandwidth management is also discussed.

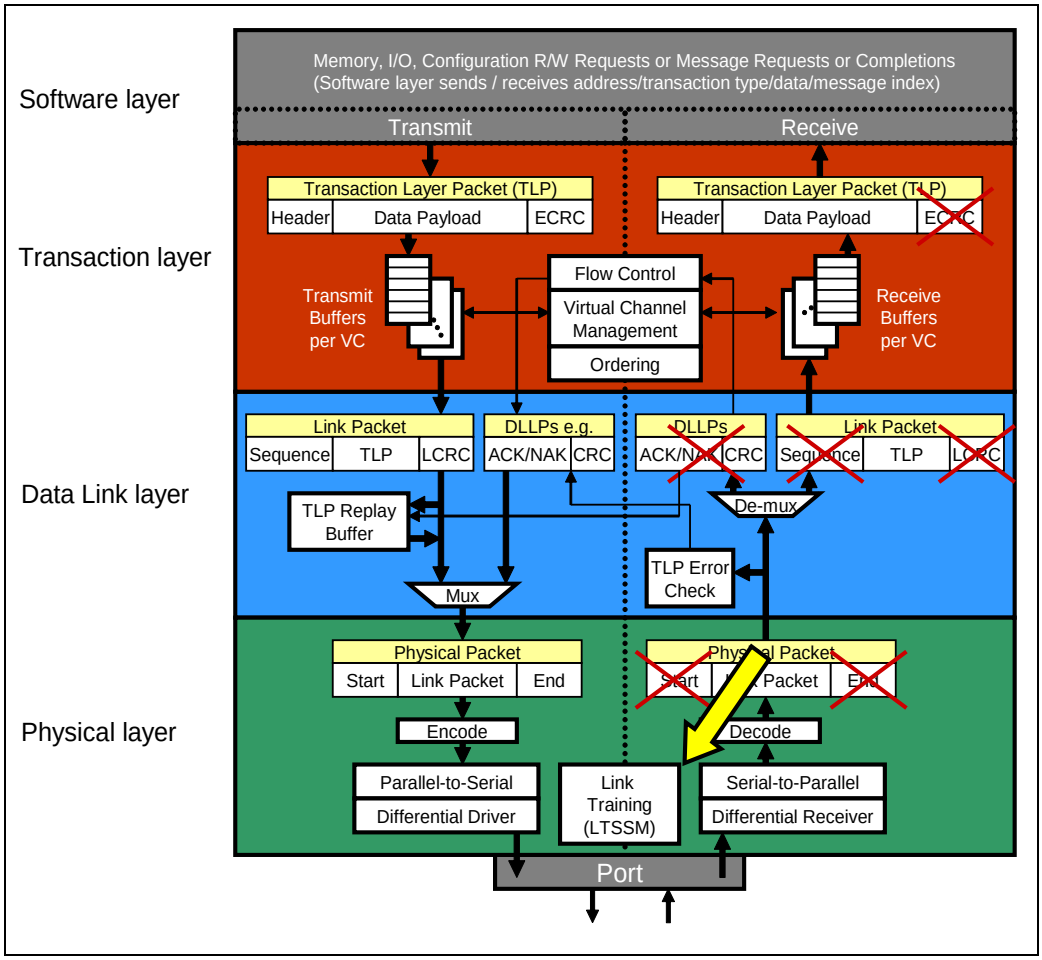
The Next Chapter

The next chapter discusses error types that occur in a PCIe Port or Link, how they are detected, reported, and options for handling them. Since PCIe is designed to be backward compatible with PCI error reporting, a review of the PCI approach to error handling is included as background information. Then we focus on PCIe error handling of correctable, non-fatal and fatal errors.

Overview

Link initialization and training is a hardware-based (not software) process controlled by the Physical Layer. The process configures and initializes a device's link and port so that normal packet traffic proceeds on the link.

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



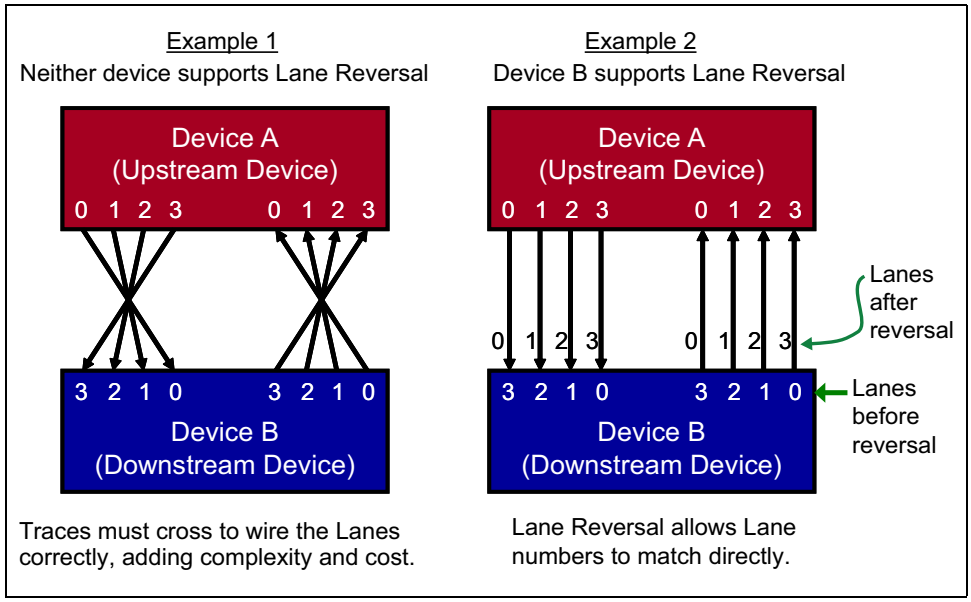
The full training process is automatically initiated by hardware after a reset and is managed by the LTSSM (Link Training and Status State Machine), shown in Figure 14-1 on page 506.

Several things are configured during the Link initialization and training process. Let's consider what they are and define some terms up front.

- **Bit Lock**: When Link training begins the Receiver's clock is not yet synchronized with the transmit clock of the incoming signal, and is unable to reliably sample incoming bits. During Link training, the Receiver CDR (Clock and Data Recovery) logic recreates the Transmitter's clock by using the incoming bit stream as a clock reference. Once the clock has been recovered from the stream, the Receiver is said to have acquired Bit Lock and is then able to sample the incoming bits. For more on the Bit Lock mechanism, see "Achieving Bit Lock" on page 395.
- **Symbol Lock**: For 8b/10b encoding (used in Gen1 and Gen2), the next step is to acquire Symbol Lock. This is a similar problem in that the receiver can now see individual bits but doesn't know where the boundaries of the 10-bit Symbols are found. As TS1s and TS2s are exchanged, Receivers search for a recognizable pattern in the bit stream. A simple one to use for this is the COM Symbol. Its unique encoding makes it easy to recognize and its arrival shows the boundary of both the Symbol and the Ordered Set since a TS1 or TS2 must be in progress. For more on this, see "Achieving Symbol Lock" on page 396.
- **Block Lock**: For 8.0 GT/s (Gen3), the process is a little different from Symbol Lock because since 8b/10b encoding is not used, there are no COM characters. However, Receivers still need to find a recognizable packet boundary in the incoming bit stream. The solution is to include more instances of the EIEOS (Electrical Idle Exit Ordered Set) in the training sequence and use that to locate the boundaries. An EIEOS is recognizable as a pattern of alternating 00h and FFh bytes, and it defines the Block boundary because, by definition, when that pattern ends the next Block must begin.
- **Link Width**: Devices with multiple Lanes may be able to use different Link widths. For example, a device with a x2 port may be connected to one with a x4 port. During Link training, the Physical Layer of both devices tests the Link and sets the width to the highest common value.
- **Lane Reversal:** The Lanes on a multi-Lane device's port are numbered sequentially beginning with Lane 0. Normally, Lane 0 of one device's port connects to Lane 0 of the neighbor's port, Lane 1 to Lane 1, and so on. However, sometimes it's desirable to be able to logically reverse the Lane numbers to simplify routing and allow the Lanes to be wired directly without having to crisscross (see Figure 14-2 on page 508). As long as one device supports the optional Lane Reversal feature, this will work. The situation is detected dur-

ing Link training and one device must internally reverse its Lane numbering. Since the spec doesn't require support for this, board designers will need to verify that at least one of the connected devices supports this feature before wiring the Lanes in reverse order.

Figure 14-2: Lane Reversal Example (Support Optional)



- **Polarity Inversion**: The D+ and D- differential pair terminals for two devices may also be reversed as needed to make board layout and routing easier. Every Receiver Lane must independently check for this and automatically correct it as needed during training, as illustrated in Figure 14-3 on page 509. To do this, the Receiver looks at Symbols 6 to 15 of the incoming TS1s or TS2s. If a D21.5 is received instead of a D10.2 in a TS1, or a D26.5 instead of the D5.2 expected for a TS2, then the polarity of that lane is inverted and must be corrected. Unlike Lane reversal, support for this feature is mandatory.

15 Error Detection and Handling

The Previous Chapter

This chapter describes the operation of the Link Training and Status State Machine (LTSSM) of the Physical Layer. The initialization process of the Link is described from Power-On or Reset until the Link reaches fully-operational L0 state during which normal packet traffic occurs. In addition, the Link power management states L0s, L1, L2, and L3 are discussed along with the state transitions. The Recovery state, during which bit lock, symbol lock or block lock are re-established is described. Link speed and width change for Link bandwidth management is also discussed.

This Chapter

Although care is always taken to minimize errors they can't be eliminated, so detecting and reporting them is an important consideration. This chapter discusses error types that occur in a PCIe Port or Link, how they are detected, reported, and options for handling them. Since PCIe is designed to be backward compatible with PCI error reporting, a review of the PCI approach to error handling is included as background information. Then we focus on PCIe error handling of correctable, non-fatal and fatal errors.

The Next Chapter

The next chapter provides an overall context for the discussion of system power management and a detailed description of PCIe power management, which is compatible with the *PCI Bus PM Interface Spec* and the *Advanced Configuration and Power Interface* (ACPI). PCIe defines extensions to the PCI-PM spec that focus primarily on Link Power and event management.

Background

Software backward compatibility with PCI is an important feature of PCIe, and that's accomplished by retaining the PCI configuration registers that were already in place. PCI verified the correct parity on each transmission phase of the bus to check for errors. Detected errors were recorded in the Status register and could optionally be reported with either of two side-band signals: PERR# (Parity Error) for a potentially recoverable parity fault during data transmission, and SERR# (System Error) for a more serious problem that was usually not recoverable. These two types can be categorized as follows:

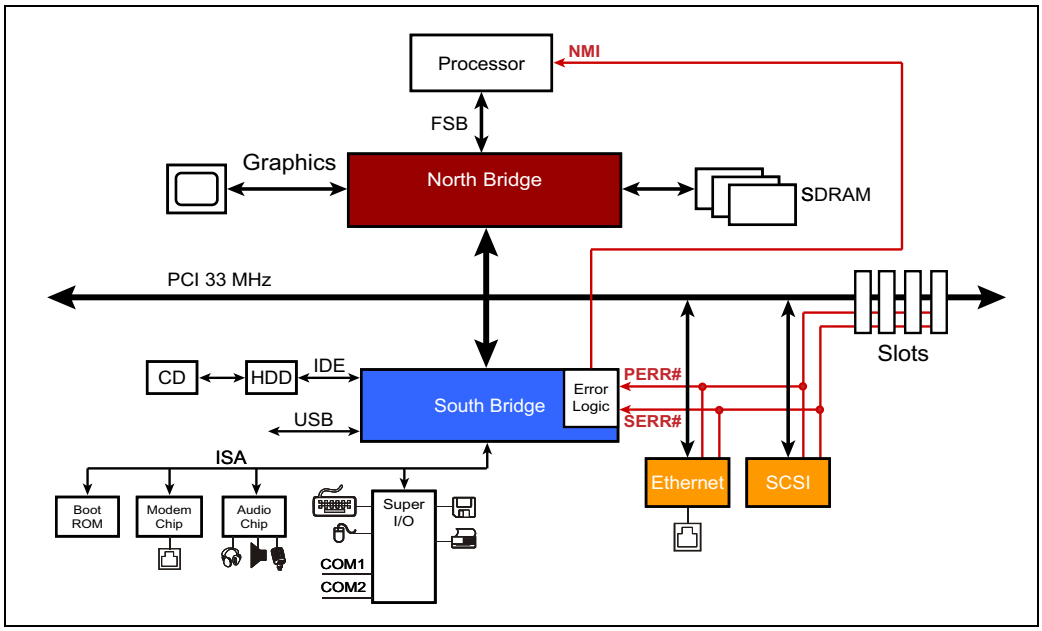
- Ordinary data parity errors — reported via PERR#
- Data parity errors during multi-task transactions (special cycles) — reported via SERR#
- Address and command parity errors — reported via SERR#
- Other types of errors (device-specific) — reported via SERR#

How the errors should be handled was outside the scope of the PCI spec and might include hardware support or device-specific software. As an example, a data parity error on a read from memory might be recovered in hardware by detecting the condition and simply repeating the Request. That would be a safe step if the memory contents weren't changed by the failed operation.

As shown in Figure 15-1 on page 649, both error pins were typically connected to the chipset and used to signal the CPU in a consumer PC. These machines were very cost sensitive, so they didn't usually have the budget for much in the way of error handling. Consequently, the resulting error reporting signal chosen was the NMI (Non-Maskable Interrupt) signal from the chipset to the processor that indicated significant system trouble requiring immediate attention. Most consumer PCs didn't include an error handler for this condition, so the system would simply be stopped to avoid corruption and the BSOD (Blue Screen Of Death) would inform the operator. An example of an SERR# condition would be an address parity mismatch seen during the command phase of a transaction. This is a potentially destructive case because the wrong target might respond. If that happened and SERR# reported it, recovery would be difficult and would probably require significant software overhead. (To learn more about PCI error handling, refer to MindShare's book *PCI System Architecture*.)

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

Figure 15-1: PCI Error Handling



PCI-X 2.0 uses source-synchronous clocking to achieve faster data rates (up to 4GB/s). This bus targeted high-end enterprise systems because it was generally too expensive for consumer machines. Since these high-performance systems also require high availability, the spec writers chose to improve the error handling by adding Error-Correcting Code (ECC) support. ECC allows more robust error detection and enables correction of single-bit errors on the fly. ECC is very helpful in minimizing the impact of transmission errors. (To learn more about PCI-X error handling, see MindShare's book *PCI-X System Architecture*.)

PCIe maintains backward compatibility with these legacy mechanisms by using the error status bits in the legacy configuration registers to record error events in PCIe that are analogous to those of PCI. That lets legacy software see PCIe error events in terms that it understands, and allows it to operate with PCIe hardware. See "PCI-Compatible Error Reporting Mechanisms" on page 674 for the details of these registers.

PCIe Error Definitions

The spec uses four general terms regarding errors, defined here:

1. **Error Detection** - the process of determining that an error exists. Errors are discovered by an agent as a result of a local problem, such as receiving a bad packet, or because it received a packet signaling an error from another device (like a poisoned packet).
2. **Error Logging** - setting the appropriate bits in the architected registers based on the error detected as an aid for error-handling software.
3. **Error Reporting** - notifying the system that an error condition exists. This can take the form of an error Message being delivered to the Root Complex, assuming the device is enabled to send error messages. The Root, in turn, can send an interrupt to the system when it receives an error Message.
4. **Error Signaling** - the process of one agent notifying another of an error condition by sending an error Message, or sending a Completion with a UR (Unsupported Request) or CA (Completer Abort) status, or poisoning a TLP (also known as error forwarding).

PCIe Error Reporting

Two error reporting levels are defined for PCIe. The first is a Baseline capability required for all devices. This includes support for legacy error reporting as well as basic support for reporting PCIe errors. The second is an optional Advanced Error Reporting Capability that adds a new set of configuration registers and tracks many more details about which errors have occurred, how serious they are and in some cases, can even record information about the packet that caused the error.

Baseline Error Reporting

Two sets of configuration registers are required in all devices in support of Baseline error reporting. These are described in detail in “Baseline Error Detection and Handling” on page 674 and are summarized here:

- PCI-compatible Registers — these are the same registers used by PCI and provide backward compatibility for existing PCI-compatible software. To make this work, PCIe errors are mapped to PCI-compatible errors, making them visible to the legacy software.

16 Power Management

The Previous Chapter

The previous chapter discusses error types that occur in a PCIe Port or Link, how they are detected, reported, and options for handling them. Since PCIe is designed to be backward compatible with PCI error reporting, a review of the PCI approach to error handling is included as background information. Then we focus on PCIe error handling of correctable, non-fatal and fatal errors.

This Chapter

This chapter provides an overall context for the discussion of system power management and a detailed description of PCIe power management, which is compatible with the *PCI Bus PM Interface Spec* and the *Advanced Configuration and Power Interface* (ACPI). PCIe defines extensions to the PCI-PM spec that focus primarily on Link Power and event management. An overview of the OnNow Initiative, ACPI, and the involvement of the Windows OS is also provided.

The Next Chapter

The next chapter details the different ways that PCIe Functions can generate interrupts. The old PCI model used pins for this, but side-band signals are undesirable in a serial model so support for the in-band MSI (Message-Signaled Interrupts) mechanism was made mandatory. The PCI INTx# pin operation can still be emulated in support of a legacy system using PCIe INTx messages. Both the PCI legacy INTx# method and the newer versions of MSI/MSI-X are described.

Introduction

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

- **PCI-Compatible PM**. PCIe power management is hardware and software compatible with the PCI-PM and ACPI specs. This support requires that all Functions include the PCI Power Management Capability registers, allowing software to transition a Function between PM states under software control through the use of Configuration requests. This was modified in the 2.1 spec revision with the addition of Dynamic Power Allocation (DPA), another set of registers that added several substates to the D0 power state to give software a finer-grained PM mechanism.
- **Native PCIe Extensions**. These define autonomous, hardware-based Active State Power Management (ASPM) for the Link, as well as mechanisms for waking the system, a Message transaction to report Power Management Events (PME), and a method for calculating and reporting the low-power-to-active-state latency.
- **Bandwidth Management.** The 2.1 spec revision added the ability for hardware to automatically change either the Link width or Link data rate or both to improve power consumption. This allows high performance when needed and keeps power usage low when lower performance is acceptable. Even though Bandwidth Management is considered a Power Management topic, we describe this capability in the section "Dynamic Bandwidth Changes" on page 618 in the "Link Initialization & Training" chapter because it involves the LTSSM.
- **Event Timing Optimization.** Peripheral devices that initiate bus master events or interrupts without regard to the system power state cause other system components to stay in high power states to service them, resulting in higher power consumption than would be necessary. This shortcoming was corrected in the 2.1 spec by adding two new mechanisms: Optimized Buffer Flush and Fill (OBFF), which lets the system inform peripherals about the current system power state, and Latency Tolerance Reporting (LTR), which allows devices to report the service delay they can tolerate at the moment.

This chapter is segmented into several major sections:

1. The first part is a primer on power management in general and covers the role of system software in controlling power management features. This discussion only considers the Windows Operating System perspective since it's the most common one for PCs, and other OSs are not described.

2. The second section, "Function Power Management" on page 713, discusses the method for putting Functions into their low-power device states using the PCI-PM capability registers. Note that some of the register definitions are modified or unused by PCIe Functions.
3. "Active State Power Management (ASPM)" on page 735 describes the hardware-based autonomous Link power management. Software determines which level of ASPM to enable for the environment, possibly by reading the recovery latency values that will be incurred for that Function, but after that the timing of the power transitions is controlled by hardware. Software doesn't control the transitions and is unable to see which power state the Link is in.
4. "Software Initiated Link Power Management" on page 760 discusses the Link power management that is forced when software changes the power state of a device.
5. "Link Wake Protocol and PME Generation" on page 768 describes how Devices may request that software return them to the active state so they can service an event. When power has been removed from a Device, auxiliary power must be present if it is to monitor events and signal a Wakeup to the system to get power restored and reactivate the Link.
6. Finally, event-timing features are described, including OBFF and LTR.

Power Management Primer

The *PCI Bus PM Interface spec* describes the power management registers required for PCIe. These permit the OS to manage the power environment of a Function directly. Rather than dive into a detailed description, let's start by describing where this capability fits in the overall context of the system.

Basics of PCI PM

This section provides an overview of how a Windows 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 706 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 dictate the PM policies that the OS uses. They do, however, define the registers (and some data structures) that are used to control the power usage of a Function.

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

Element	Responsibility
OS	Directs **overall system power management** by sending requests to the ACPI Driver, device driver, and the PCI Express Bus Driver. Applications that are power conservation-aware interact with the OS to accomplish device power management.
ACPI Driver	Manages configuration, power management, and thermal control of embedded system devices that don't adhere to an industry-standard spec. Examples of this include chipset-specific registers, system board-specific registers to control power planes, etc. The PM registers within PCIe 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).
Device Driver	The **Class driver** 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 vendor means that it doesn't have 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 device driver also doesn't understand device characteristics that are peculiar to a specific bus implementation of that device type. As an example, it won't understand a PCIe Function's configuration register set. The **PCI Express Bus Driver** is the one to communicate with those registers. When it receives requests from the OS to control the power state of a PCIe 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 device driver saves the contents of its associated Function's device-specific registers (in other words, 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, the device driver 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 device driver then restores the context to the Function's device-specific registers.
Miniport Driver	**Supplied by the vendor of a device**, it receives requests from the Class driver and converts them into the proper series of accesses to the device's register set.

17 Interrupt Support

The Previous Chapter

The previous chapter provides an overall context for the discussion of system power management and a detailed description of PCIe power management, which is compatible with the *PCI Bus PM Interface Spec* and the *Advanced Configuration and Power Interface* (ACPI) spec. PCIe defines extensions to the PCI-PM spec that focus primarily on Link Power and event management. An overview of the OnNow Initiative, ACPI, and the involvement of the Windows OS is also provided.

This Chapter

This chapter describes the different ways that PCIe Functions can generate interrupts. The old PCI model used pins for this, but sideband signals are undesirable in a serial model so support for the inband MSI (Message Signaled Interrupt) mechanism was made mandatory. The PCI INTx# pin operation can still be emulated using PCIe INTx messages for software backward compatibility reasons. Both the PCI legacy INTx# method and the newer versions of MSI/MSI-X are described.

The Next Chapter

The next chapter describes three types of resets defined for PCIe: Fundamental reset (consisting of cold and warm reset), hot reset, and function-level reset (FLR). The use of a sideband reset PERST# signal to generate a system reset is discussed, and so is the inband TS1 based Hot Reset described.

Interrupt Support Background

General

The PCI architecture supported interrupts from peripheral devices as a means of improving their performance and offloading the CPU from the need to poll devices to determine when they require servicing. PCIe inherits this support largely unchanged from PCI, allowing software backwards compatibility to PCI. We provide a background to system interrupt handling in this chapter, but the reader who wants more details on interrupts is encouraged to look into these references:

- For PCI interrupt background, refer to the PCI spec rev 3.0 or to chapter 14 of MindShare's textbook: PCI System Architecture (www.mindshare.com).
- To learn more about Local and IO APICs, refer to MindShare's textbook: x86 Instruction Set Architecture.

Two Methods of Interrupt Delivery

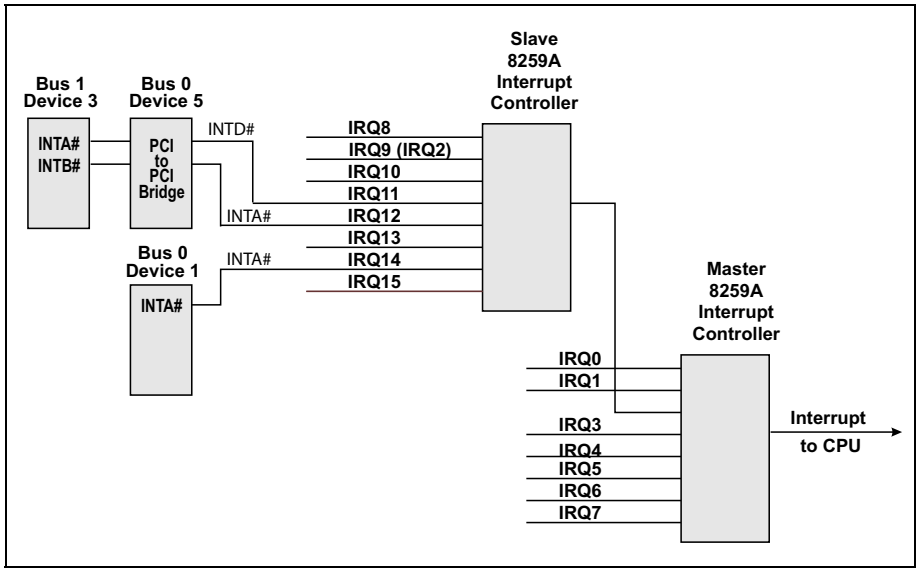
PCI used sideband interrupt wires that were routed to a central interrupt controller. This method worked well in simple, single-CPU systems, but had some shortcomings that motivated moving to a newer method called MSI (Message Signaled Interrupts) with an extension called MSI-X (eXtented).

Legacy PCI Interrupt Delivery — This original mechanism defined for the PCI bus consists of up to four signals per device or INTx# (INTA#, INTB#, INTC#, and INTD#) as shown in Figure 17-1 on page 795. In this model, the pins are shared by wire-ORing them together, and they'd eventually be connected to an input on the 8259 PIC (Programmable Interrupt Controller). When a pin is asserted, the PIC in turn asserts its interrupt request pin to the CPU as part of a process described in "The Legacy Model" on page 796.

PCIe supports this PCI interrupt functionality for backward compatibility, but a design goal for serial transports is to minimize the pin count. As a result, the INTx# signals were not implemented as sideband pins. Instead, a Function can generate an inband interrupt message packet to indicate the assertion or deassertion of a pin. These messages act as "virtual wires", and target the interrupt controller in the system (typically in the Root Complex), as shown in Figure 17-2 on page 796. This picture also illustrates how an older PCI device using the

pins can work in a PCIe system; the bridge translates the assertion of a pin into an interrupt emulation message (INTx) going upstream to the Root Complex. The expectation is that PCIe devices would not normally need to use the INTx messages but, at the time of this writing, in practice they often do because system software has not been updated to support MSI.

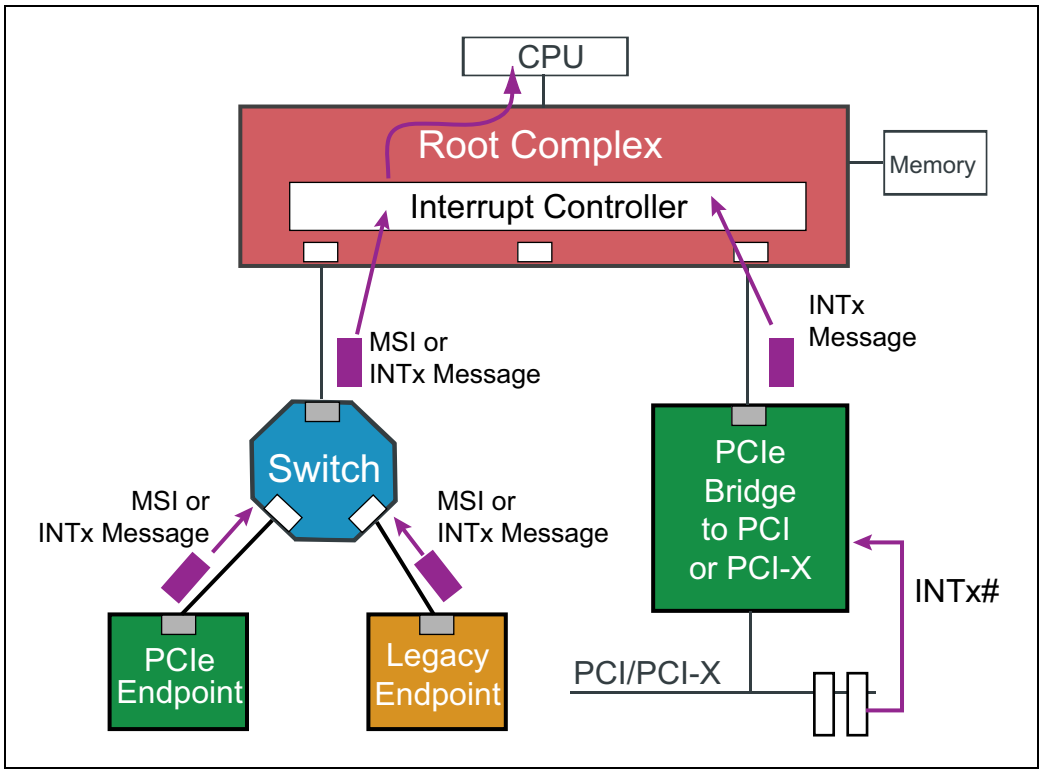
Figure 17-1: PCI Interrupt Delivery



MSI Interrupt Delivery — MSI eliminates the need for sideband signals by using memory writes to deliver the interrupt notification. The term "Message Signaled Interrupt" can be confusing because its name includes the term "Message" which is a type of TLP in PCIe, but an MSI interrupt is a Posted Memory Write instead of a Message transaction. MSI memory writes are distinguished from other memory writes only by the addresses they target, which are typically reserved by the system for interrupt delivery (e.g., x86-based systems traditionally reserve the address range FEEx_xxxxh for interrupt delivery).

Figure 17-2 illustrates the delivery of interrupts from various types of PCIe devices. All PCIe devices are required to support MSI, but software may or may not support MSI, in which case, the INTx messages would be used. Figure 17-2 also shows how a PCIe-to-PCI Bridge is required to convert sideband interrupts from connected PCI devices to PCIe-supported INTx messages.

Figure 17-2: Interrupt Delivery Options in PCIe System



The Legacy Model

General

To illustrate the legacy interrupt delivery model, refer to Figure 17-3 on page 797 and consider the usual steps involved in interrupt delivery using the legacy method of interrupt pins:

1. The device generates an interrupt by asserting its pin to the controller. In older systems this controller was typically an Intel 8259 PIC that had 15 IRQ inputs and one INTR output. The PIC would then assert INTR to inform the CPU that one or more interrupts were pending.

18 System Reset

The Previous Chapter

The previous chapter describes the different ways that PCIe Functions can generate interrupts. The old PCI model used pins for this, but sideband signals are undesirable in a serial model so support for the inband MSI (Message Signaled Interrupt) mechanism was made mandatory. The PCI INTx# pin operation can still be emulated using PCIe INTx messages for software backward compatibility reasons. Both the PCI legacy INTx# method and the newer versions of MSI/MSI-X are described.

This Chapter

This chapter describes the four types of resets defined for PCIe: cold reset, warm reset, hot reset, and function-level reset. The use of a side-band reset PERST# signal to generate a system reset is discussed, and so is the in-band TS1 used to generate a Hot Reset.

The Next Chapter

The next chapter describes the PCI Express hot plug model. A standard usage model is also defined for all devices and form factors that support hot plug capability. Power is an issue for hot plug cards, too, and when a new card is added to a system during runtime, it's important to ensure that its power needs don't exceed what the system can deliver. A mechanism was needed to query and control the power requirements of a device, Power Budgeting provides this.

Two Categories of System Reset

The PCI Express spec describes four types of reset mechanisms. Three of these were part of the earlier revisions of the PCIe spec and are collectively referred to now as **Conventional Resets,** and two of them are called Fundamental Resets. The fourth category and method, added with the 2.0 spec revision, is called the **Function Level Reset**.

Conventional Reset

Fundamental Reset

A Fundamental Reset is handled in hardware and resets the entire device, re-initializing every state machine and all the hardware logic, port states and configuration registers. The exception to this rule is a group of some configuration register fields that are identified as "sticky", meaning they retain their contents unless all power is removed. This makes them very useful for diagnosing problems that require a reset to get a Link working again, because the error status survives the reset and is available to software afterwards. If main power is removed but Vaux is available, that will also maintain the sticky bits, but if both main power and Vaux are lost, the sticky bits will be reset along with everything else.

A Fundamental Reset will occur on a system-wide reset, but it can also be done for individual devices.

Two types of Fundamental Reset are defined:

- **Cold Reset**: The result when the main power is turned on for a device. Cycling the power will cause a cold reset.
- **Warm Reset (optional):** Triggered by a system-specific means without shutting off main power. For example, a change in the system power status might be used to initiate this. The mechanism for generating a Warm Reset is not defined by the spec, so the system designer will choose how this is done.

When a Fundamental Reset occurs:

- For positive voltages, receiver terminations are required to meet the $Z_{RX\text{-}HIGH\text{-}IMP\text{-}DC\text{-}POS}$ parameter. At 2.5 GT/s, this is no less than 10 KΩ. At the higher speeds it must be no less than 10 KΩ for voltages below 200mv, and 20 KΩ for voltages above 200mv. These are the values when the terminations are not powered.
- Similarly for negative voltages, the $Z_{RX\text{-}HIGH\text{-}IMP\text{-}DC\text{-}NEG}$ parameter, the value is a minimum of 1 KΩ in every case.
- Transmitter terminations are required to meet the output impedance $Z_{TX\text{-}DIFF\text{-}DC}$ from 80 to 120Ω for Gen1 and max of 120Ω for Gen2 and Gen3, but may place the driver in a high impedance state.
- The transmitter holds a DC common mode voltage between 0 and 3.6 V.

When exiting from a Fundamental Reset:

- The receiver single-ended terminations must be present when receiver terminations are enabled so that Receiver Detect works properly (40-60Ω for Gen1 and Gen2, and 50Ω +/– 20% for Gen3. By the time Detect is entered, the common-mode impedance must be within the proper range of 50Ω +/– 20%.
- must re-enable its receiver terminations $Z_{RX\text{-}DIFF\text{-}DC}$ of 100Ω within 5 ms of Fundamental Reset exit, making it detectable by the neighbor's transmitter during training.
- The transmitter holds a DC common mode voltage between 0 and 3.6 V.

Two methods of delivering a Fundamental Reset are defined. First, it can be signaled with an auxiliary side-band signal called PERST# (PCI Express Reset). Second, when PERST# is not provided to an add-in card or component, a Fundamental Reset is generated autonomously by the component or add-in card when the power is cycled.

PERST# Fundamental Reset Generation

A central resource device such as a chipset in the PCI Express system provides this reset. For example, the IO Controller Hub (ICH) chip in Figure 18-1 on page 836 may generate PERST# based on the status of the system power supply 'POWERGOOD' signal, since this indicates that the main power is turned on and stable. If power is cycled off, POWERGOOD toggles and causes PERST# to assert and deassert., resulting in a Cold Reset. The system may also provide a method of toggling PERST# by some other means to accomplish a 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 do so. PERST# also feeds the PCIe-to-PCI-X bridge shown in the figure. Bridges always forward a reset on their primary (upstream) bus to their secondary (downstream) bus, so the PCI-X bus sees RST# asserted.

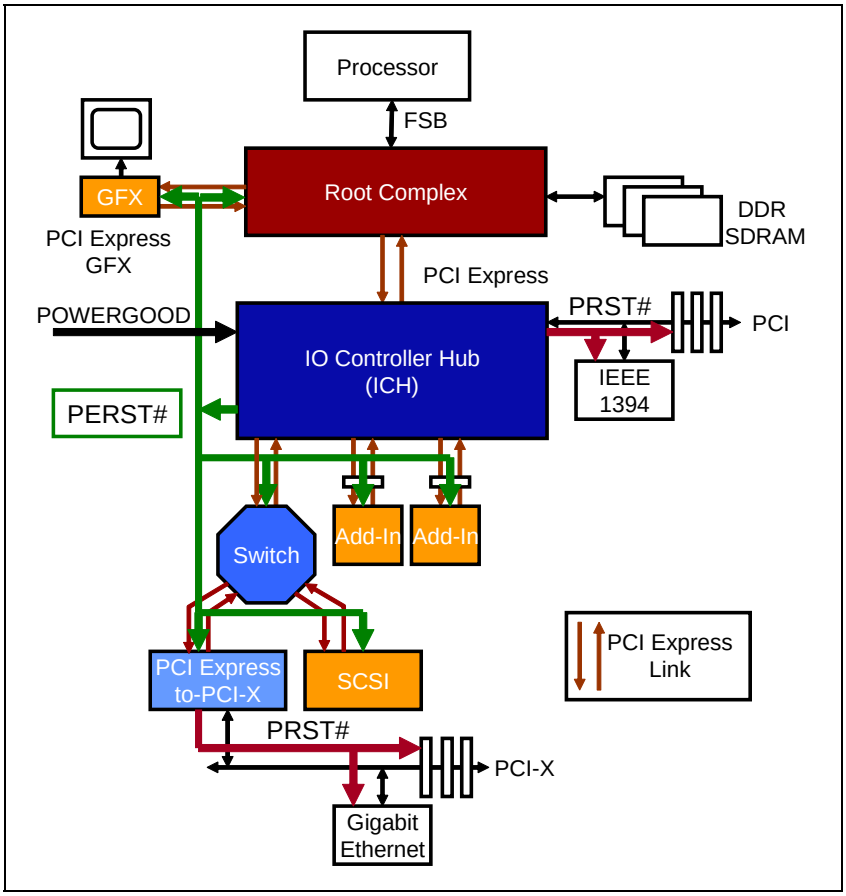
Autonomous Reset Generation

A device must be designed to generate its own reset in hardware upon application of main power. The spec doesn't describe how this would be done, so a self-reset mechanism can be built into the device or added as external logic. For example, an add-in card that detects Power-On may use that event to generate a local reset to its device. The device must also generate an autonomous reset if it detects its power go outside of the limits specified.

Link Wakeup from L2 Low Power State

As an example of the need for an autonomous reset, a device whose main power has been turned off as part of a power management policy may be able to request a return to full power if it was designed to signal a wakeup. When power is restored, the device must be reset. The power controller for the system may assert the PERST# pin to the device, as shown in Figure 18-1 on page 836, but if it doesn't, or if the device doesn't support PERST#, the device must autonomously generate its own Fundamental Reset when it senses main power re-applied.

Figure 18-1: PERST# Generation



19 Hot Plug and Power Budgeting

The Previous Chapter

The previous chapter describes three types of resets defined for PCIe: Fundamental reset (consisting of cold and warm reset), hot reset, and function-level reset (FLR). The use of a side-band reset PERST# signal to generate a system reset is discussed, and so is the in-band TS1 based Hot Reset described.

This Chapter

This chapter describes the PCI Express hot plug model. A standard usage model is also defined for all devices and form factors that support hot plug capability. Power is an issue for hot plug cards, too, and when a new card is added to a system during runtime, it's important to ensure that its power needs don't exceed what the system can deliver. A mechanism was needed to query the power requirements of a device before giving it permission to operate. Power budgeting registers provide that.

The Next Chapter

The next chapter describes the changes and new features that were added with the 2.1 revision of the spec. Some of these topics, like the ones related to power management, are described in earlier chapters, but for others there wasn't another logical place for them. In the end, it seemed best to group them all together in one chapter to ensure that they were all covered and to help clarify what features are new.

Background

Some systems using PCIe require high availability or non-stop operation. Online service suppliers require computer systems that experience downtimes of just a few minutes a year or less. There are many aspects to building such systems, but equipment reliability is clearly important. To facilitate these goals PCIe supports the Hot Plug/Hot Swap solutions for add-in cards 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 a failed device

Prior to the widespread acceptance of PCI, many proprietary Hot Plug solutions were developed to support this type of removal and replacement of expansion cards. The original PCI implementation did not support hot removal and insertion of cards, but two standardized solutions for supporting this capability in PCI have been developed. The first is the Hot Plug PCI Card used in PC Server motherboard and expansion chassis implementations. The other is called Hot Swap and is used in CompactPCI systems based on a passive PCI backplane implementation.

In both solutions, control logic is used to electrically isolate the card logic from the shared PCI bus. Power, reset, and clock are controlled to ensure an orderly power down and power up of cards as they are removed and replaced, and status and power LEDs inform the user when it's safe to change a card.

Extending hot plug support to PCI Express cards is an obvious step, and designers have incorporated some Hot Plug features as "native" to PCIe. The spec defines configuration registers, Hot Plug Messages, and procedures to support Hot Plug solutions.

Hot Plug in the PCI Express Environment

PCIe Hot Plug is derived from the 1.0 revision of the Standard Hot Plug Controller spec (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 spec. This ensures that the PCI Express hot plug is identical from the user perspective to existing implementations based on the SHPC 1.0 spec

- Support the same software model implemented by existing operating systems. However, an OS using a SHPC 1.0 compliant driver won't work with PCI Express Hot Plug controllers because they have a different programming interface.

The registers necessary to support a Hot Plug Controller are integrated into individual Root and Switch Ports. Under Hot Plug software control, these controllers and the associated port interface must control the card interface signals to ensure orderly power down and power up as cards are changed. To accomplish that, they'll need to:

- Assert and deassert the PERST# signal to the PCI Express card connector
- Remove or apply power to the card connector.
- Selectively turn on or off the Power and Attention Indicators associated with a specific card connector to draw the user's attention to the connector and indicate whether power is applied to the slot.
- Monitor slot events (e.g. card removal) and report them 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 normally allowed to install or remove a PCI Express card without first notifying the system. Software then prepares both the card and slot and finally indicates to the operator the status of the hot plug process and notification that installation or removal may now be performed.

Surprise Removal Notification

Cards designed to the PCIe Card ElectroMechanical spec (CEM) implement card presence detect pins (PRSNT1# and PRSNT2#) on the connector. These pins are shorter than the others so that they break contact first (when the card is removed from the slot). This can be used to give advanced notice to software of a "surprise" removal, allowing time to remove power before the signals break contact.

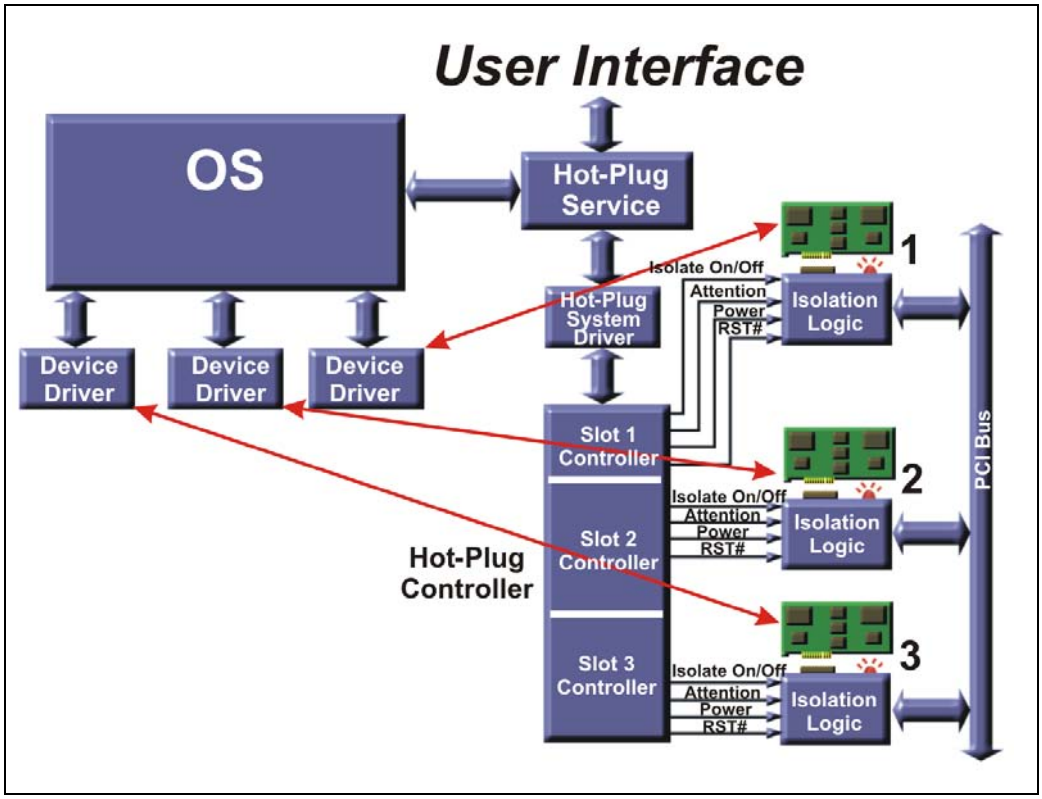
Differences between PCI and PCIe Hot Plug

The elements needed to support hot plug are essentially the same in both PCI and PCIe hot plug solutions. Figure 19-1 on page 850 shows 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 handled all the

hot plug slots on the bus. Isolation logic is needed in the PCI environment to electrically disconnect a card from the shared bus prior to making changes to avoid glitching the signals on an active bus.

PCIe uses point-to-point connections (see Figure 19-2 on page 851) that eliminate the need for isolation logic but require a separate hot plug controller for each Port to which a connector is attached. A standardized software interface defined for each Root and Switch Port controls hot plug operations.

Figure 19-1: PCI Hot Plug Elements



20 Updates for Spec Revision 2.1

Previous Chapter

The previous chapter describes the PCI Express hot plug model. A standard usage model is also defined for all devices and form factors that support hot plug capability. Power is an issue for hot plug cards, too, and when a new card is added to a system during runtime, it's important to ensure that its power needs don't exceed what the system can deliver. A mechanism was needed to query the power requirements of a device before giving it permission to operate. Power budgeting registers provide that.

This Chapter

This chapter describes the changes and new features that were added with the 2.1 revision of the spec. Some of these topics, like the ones related to power management, are described in other chapters, but for others there wasn't another logical place for them. In the end, it seemed best to group them all together in one chapter to ensure that they were all covered and to help clarify what features were new.

The Next Chapter

The next section is the book appendix which includes topics such as: Debugging PCI Express Traffic using LeCroy Tools, Markets & Applications of PCI Express Architecture, Implementing Intelligent Adapters and Multi-Host Systems with PCI Express Technology, Legacy Support for Locking and the book Glossary.

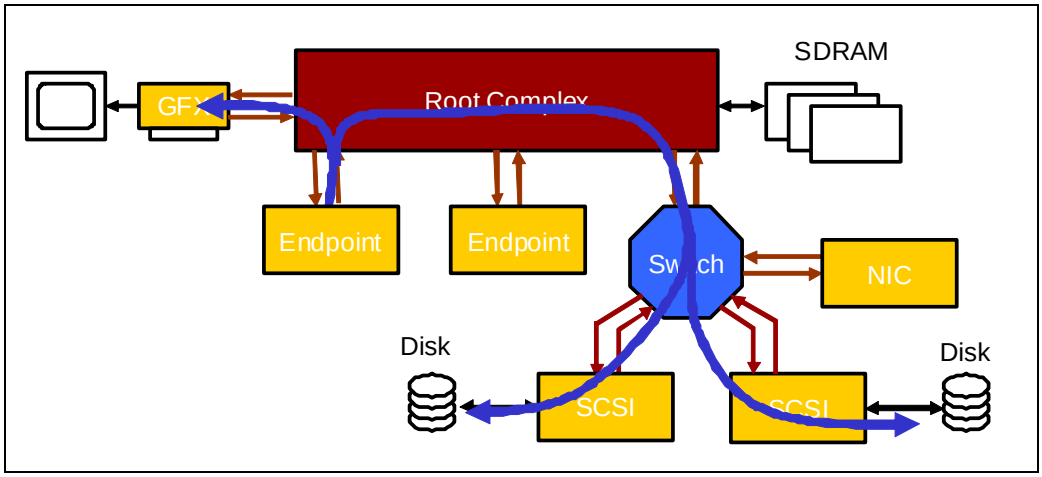
Changes for PCIe Spec Rev 2.1

The 2.1 revision of the spec for PCIe introduced several changes to enhance performance or improve operational characteristics. It did not add another data rate and that's why it was considered an incremental revision. The modifications can be grouped generally into four areas of improvement: System Redundancy, Performance, Power Management, and Configuration.

System Redundancy Improvement: Multi-casting

The Multi-casting capability allows a Posted Write TLP to be routed to more than one destination at the same time, allowing for things like automatically making redundant copies of data or supporting multi-headed graphics. As shown in Figure 20-1 on page 888, a TLP sourced from one Endpoint can be routed to multiple destinations based solely on its address. In this example, data is sent to the video port for display while redundant copies of it are automatically routed to storage. There are other ways this activity could be supported, of course, but this is very efficient in terms of Link usage since it doesn't require a recipient to re-send the packet to secondary locations.

Figure 20-1: Multicast System Example

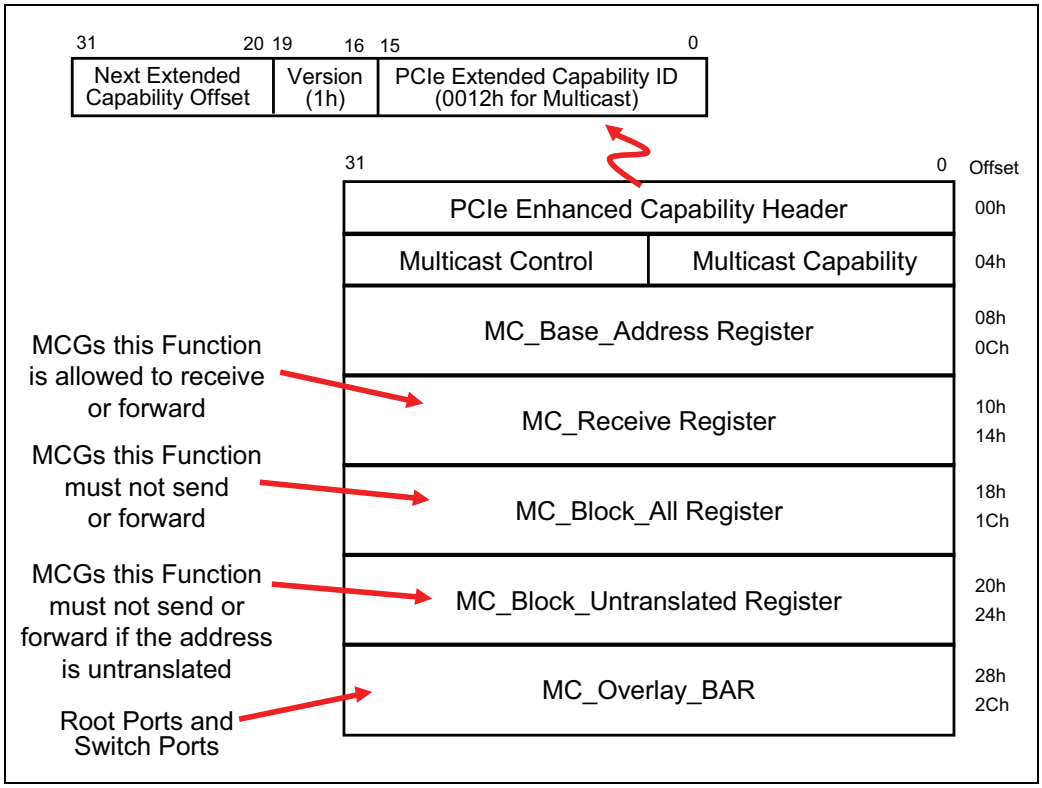


This mechanism is only supported for posted, address-routed Requests, such as Memory Writes, that contain data to be delivered and an address that can be decoded to show which Ports should receive it. Non-posted Requests will not be treated as Multicast even if their addresses fall within the MultiCast address range. Those will be treated as unicast TLPs just as they normally would.

The setup for Multicast operation involves programming a new register block for each routing element and Function that will be involved, called the Multicast Capability structure. The contents of this block are shown in Figure 20-2 on page 889, where it can be seen that they define addresses and also MCGs (MultiCast Group numbers) that explain whether a Function should send or receive copies of an incoming TLP or whether a Port should forward them. Let's

describe these registers next and discuss how they're used to create Multicast operations in a system.

Figure 20-2: Multicast Capability Registers



Multicast Capability Registers

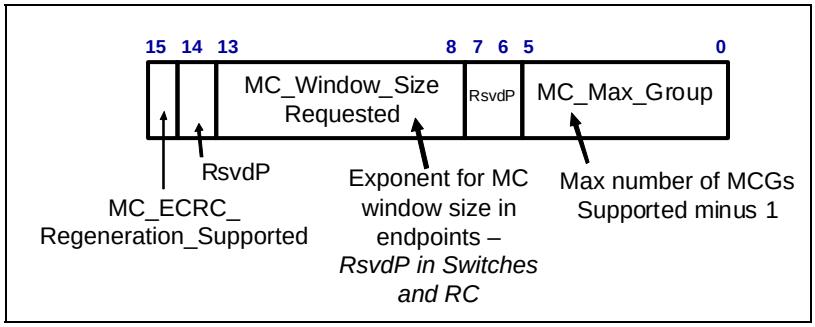
The Capability Header register at the top of the figure includes the Capability ID of 0012h, a 4-bit Version number, and a pointer to the next capability structure in the linked list of registers.

Multicast Capability

This register, shown in detail in Figure 20-3 on page 890, contains several fields. The MC_Max_Group value defines how many Multicast Groups this Function has been designed to support minus one, so that a value of zero means one

group is supported. The Window Size Requested, which is only valid for Endpoints and reserved in Switches and Root Ports, represents the address size needed for this purpose as a power of two.

Figure 20-3: Multicast Capability Register



Lastly, bit 15 indicates whether this Function supports regenerating the ECRC value in a TLP if forwarding it involved making address changes to it. Refer to the section called "Overlay Example" on page 895 for more detail on this.

Multicast Control

This register, shown in Figure 20-4 on page 890, contains the MC_Num_Group that is programmed with the number of Multicast Groups configured by software for use by this Function. The default number is zero, and the spec notes that programming a value here that is greater than the max value defined in the MC_Max_Group register will result in undefined behavior. The MC_Enable bit is used to enable the Multicast mechanism for this component.

Figure 20-4: Multicast Control Register